Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Valutazione dei path come criterio per l'esplorazione robotica

Relatore: **Prof. Francesco AMIGONI**
Correlatore: **Ing. Alberto QUATTRINI LI**

Tesi di laurea di:
Sara BIANCINI Matr. 804956
Stefano CARNOVALI Matr. 781712

Anno Accademico 2014–2015

Alla nostre famiglie che ci hanno sostenuto
in questi anni di studi...

Ringraziamenti

Desideriamo ringraziare tutte le persone che ci sono state vicine e che hanno reso possibile questo lavoro di tesi. In particolar modo un sentito ringraziamento va al relatore Professor Ing. Francesco Amigoni. Un doveroso ringraziamento va anche al correlatore Ing. Alberto Quattrini Li per l'attenzione e la pazienza con cui ci ha seguiti in questi mesi. Non può mancare il ringraziamento alle nostre famiglie a cui dedichiamo questo lavoro di tesi: senza il loro aiuto non avremmo mai raggiunto questa meta. Siamo davvero grati per tutto il sostegno economico, ma soprattutto per l'aiuto tacito o esplicito che è venuto dal loro cuore: a tutte quelle volte che ci hanno incoraggiati vedendoci alle prese con i libri, con un esame e con questo lavoro di tesi. Ci auguriamo che tutti i sacrifici spesi siano in questo modo, almeno in parte, ripagati. Infine, desideriamo ringraziare tutte quelle persone con cui abbiamo iniziato e trascorso i nostri percorsi di studi, con cui abbiamo scambiato qualche pensiero, qualche idea, qualche risata in aula. In diversi modi hanno contribuito ai nostri percorsi formativi.

Grazie a tutti.

Luglio 2015

Sommario

Il lavoro svolto in questa tesi di laurea si colloca nell'ambito della robotica mobile autonoma. In questo campo, la ricerca scientifica si occupa di sviluppare agenti robotici artificiali in grado di muoversi autonomamente in un ambiente, sia esso simulato o reale.

Il sistema proposto in questa tesi si riferisce principalmente ad un contesto di Urban Search And Rescue (USAR). In questo tipo di situazioni, generate in seguito a disastri naturali o incidenti, si rende necessario cercare eventuali vittime o superstiti all'interno dell'ambiente colpito da terremoti, incendi, crolli di abitazioni o altri disastri. In questi contesti, l'utilizzo di agenti robotici autonomi o semi-autonomi può portare a numerosi benefici, eliminando o limitando i rischi legati ad un intervento diretto dei soccorritori e salvaguardando la loro vita.

Lo scopo di questo lavoro di tesi è l'implementazione di un algoritmo per l'esplorazione di ambienti interni con singolo agente autonomo, che valuta i percorsi per raggiungere le prossime posizioni di osservazione e non solo quest'ultime, come avviene usualmente in letteratura. Questo permette al robot di scegliere tra i percorsi possibili quello che massimizza le informazioni ottenibili durante il movimento. La scelta viene effettuata tenendo in considerazione diversi criteri e cercando di bilanciare fattori come la lunghezza del percorso, il consumo della batteria e l'area percepibile lungo il percorso stesso.

I risultati sperimentali ottenuti sembrano dimostrare che l'algoritmo di esplorazione dai noi implementato può portare a dei benefici in alcune tipologie di ambienti, in particolare in quelli caratterizzati da grandi stanze con la presenza di numerosi ostacoli che rendono difficile la percezione completa di tali stanze da un unico punto al loro interno.

Abstract

The work presented in this thesis is placed in the area of autonomous mobile robotics. In this field, scientific research deals with the development of artificial agents able to move in an environment, either simulated or real, independently from human users.

The system proposed in this thesis can be applied in a context of Urban Search And Rescue (USAR). In these situations, encountered after natural disasters or accidents, it is necessary to look for any survivors or victims inside environments affected by earthquakes, fires, collapses, or others disasters. In such a case, the use of autonomous or semi-autonomous robotic agents can lead to numerous benefits by removing or limiting the risks associated with the direct intervention of human rescuers, safeguarding their lives.

The purpose of this thesis is the design and implementation of an algorithm for the exploration of indoor environments with a single autonomous agent, that focuses on the generation and evaluation of the possible paths to reach the next observation positions. This contrasts with the mainstream approach in the literature that only considers the evaluation of the observation positions. Our approach allows the robot to choose between the possible paths the one that maximizes the information that is expected to be obtained during the movement. The choice is made by taking into account different criteria and trying to balance factors such as the length of the path, the battery consumption, and the perceived area.

The experimental results suggest that our proposed exploration algorithm can lead to benefits in some types of environments, especially in those characterized by large rooms with the presence of several obstacles which prevent the complete view of the rooms without moving inside them.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli algoritmi

Capitolo 1

Introduzione

Fin dalla sua nascita la robotica autonoma ha cercato di imitare il comportamento umano e realizzare macchine che emulassero la nostra capacità di ragionamento e di interazione con l'ambiente. Questo obiettivo è ben lontano dall'essere raggiunto e ancora oggi la ricerca si concentra maggiormente sullo studio e la risoluzione dei sotto-problemi collegati a questa visione. Uno dei problemi più ricorrenti della robotica autonoma è la decisione sul movimento dei robot. Un robot inserito in un ambiente sconosciuto deve essere in grado di costruirne una rappresentazione (mappa) e orientarsi al fine di raggiungere un determinato obiettivo con la massima efficienza possibile. Per fare questo è necessario che il robot percepisca l'ambiente circostante estraendo tutte le informazioni necessarie per completare il proprio compito. In base allo scopo da raggiungere, l'efficienza può essere misurata con diverse metriche a volte contrastanti fra loro (per esempio: il tempo impiegato o la qualità della mappa).

Uno degli scenari possibili dell'utilizzo di robot autonomi è la ricerca di vittime coinvolte in incidenti o disastri naturali in cui l'intervento diretto umano sarebbe troppo pericoloso. Nel caso di ambienti indoor si parla generalmente di Urban Search And Rescue (USAR) in cui l'esplorazione avviene in ambienti strutturati ma parzialmente o totalmente sconosciuti, almeno inizialmente. L'assenza o la scarsità di informazioni sulla posizione o presenza di eventuali vittime permette di ricondurre il problema alla massimizzazione dell'area esplorata. Questo obiettivo può essere raggiunto, da parte dei robot coinvolti nell'esplorazione, con una scelta accurata della prossima destinazione, in modo da aumentare il più possibile l'area mappata nel poco tempo a disposizione.

Lo scopo della tesi è implementare un algoritmo per permettere a un singolo robot l'esplorazione di ambienti interni che, diversamente dalle soluzioni più utilizzate fino ad ora, non tenga conto solo della destinazione scelta, ma anche del percorso da seguire per raggiungerla. Questo permette al robot di scegliere tra i percorsi possibili quello che massimizza le informazioni che è possibile ottenere durante il movimento. La scelta viene effettuata tenendo in considerazione diversi criteri e cercando di bilanciare fattori come la lunghezza del percorso, il consumo della batteria, l'area potenzialmente percepibile lungo il percorso.

Come in molti dei lavori precedenti in questo ambito, il nostro approccio sfrutta il concetto di frontiera, introdotto da Yamauchi [23]. Le frontiere sono regioni al confine tra lo spazio conosciuto e lo spazio inesplorato, e la scelta del prossimo punto da visitare in queste aree permette di concentrare l'attenzione del robot nelle zone più promettenti della mappa. Numerose strategie di esplorazione sono state proposte dai ricercatori, ognuna con i propri vantaggi e svantaggi; molte di queste sono state presentate e messe a confronto in alcune pubblicazioni [2]. Molte di queste strategie si basano però su approcci ad hoc che non garantiscono un elevato grado di flessibilità. Per ovviare a questo problema è stato proposto il framework Multi-Criteria-Decision-Making (MCDM) [8], una soluzione con solide basi matematiche e teoriche che permette la combinazione di più criteri per la valutazione delle frontiere durante l'esplorazione.

L'implementazione del nostro sistema si basa sul lavoro esistente sviluppato dal "Politecnico di Milano Autonomous Robotic Rescue Team" (PoA-ReT) [4] del Politecnico di Milano che utilizza USARSim [7, 20], un simulatore 3D orientato al soccorso robotico capace di ricreare scenari in cui testare le prestazioni di agenti robotici, autonomi o controllati da operatori umani. Questo simulatore è utilizzato nelle RoboCup Rescue Virtual Robot Competition[1].

L'ambiente viene rappresentato per mezzo di una mappa poligonale composta da segmenti che identificano muri, ostacoli e frontiere. Per poter procedere con la creazione dei percorsi verso le frontiere, si è scelto di campionare uniformemente tutta la parte già esplorata e libera da ostacoli della mappa nota a un certo istante, facendo uso dell'algoritmo Probabilistic RoadMaps (PRM) [14]. Ciò ha permesso la creazione di un grafo che comprende un sotto-insieme dei

[1] www.robocuprescue.org

percorsi possibili per muoversi all'interno dell'ambiente. Da questa struttura vengono estratti una serie di cammini verso le frontiere raggiungibili facendo uso di una versione modificata dell'algoritmo A^* chiamata $AlphA^*$ [16]. Ottenuto un insieme di percorsi si passa alla loro valutazione per mezzo del framework MCDM [8] che permette la scelta del miglior candidato, basandosi su una combinazione di criteri.

Le attività sperimentali svolte per valutare il sistema sviluppato in questo lavoro di tesi (chiamato *MCDM-Path*), hanno dato la possibilità di verificare se ci fossero miglioramenti rispetto alla strategia di riferimento utilizzata dal team PoAReT (chiamata *MCDM-Pose*) e alla strategia proposta da Tovar et al. [18] (ad oggi l'unica strategia basata sulla valutazione dell'intero path). Questi esperimenti, condotti con il simulatore USARSim, sono stati eseguiti variando, per ogni mappa utilizzata, la strategia adottata ed il punto che il robot assume come posizione di partenza della simulazione.

Le misure di performance che sono state utilizzate per la valutazione del sistema sono: l'area misurata ad un orizzonte temporale fissato, definito dal tempo medio di completamento della mappatura da parte della strategia migliore, e il tempo impiegato per mappare una certa percentuale della mappa.

I risultati ottenuti sembrano confermare un incremento delle prestazioni in determinate mappe con l'utilizzo della strategia di esplorazione MCDM-Path, grazie alla capacità di valutare l'intero percorso seguito dall'agente autonomo. I metodi basati sulla valutazione dei percorsi sembrerebbero inoltre presentare un comportamento più stabile, dovuto alla capacità di scegliere con più accuratezza il cammino da seguire, scartando il path che può sembrare più corto, ma guadagnando a lungo termine sulla distanza percorsa complessivamente, migliorando così l'efficienza.

Questa tesi non è certamente un lavoro definitivo, l'essenza stessa della ricerca scientifica, infatti, sta nel porre in discussione quanto realizzato sinora per stabilirne la correttezza e le possibilità di miglioramento. Il lavoro proposto presenta diversi spunti di ulteriore ricerca. Per quanto riguarda l'esplorazione, si possono pensare numerosi altri criteri per valutare i punti da raggiungere e i percorsi da seguire oltre a quelli utilizzati in questo lavoro, si potrebbe inoltre ipotizzare il cambiamento a run-time della strategia di esplorazione, dovuto al verificarsi di particolari eventi: cosa succede se un robot determina che la strategia utilizzata non è adeguata per esplorare l'ambiente circostante? Una delle possibilità può essere quella di cercare un modo automatico per modificare la strategia di esplorazione durante l'esecuzione. Questa ipotesi si può ricondurre

ad un'altra simile, che considera anche il tempo come una delle variabili di cui tener conto durante l'esplorazione: molto spesso può risultare utile condurre parte dell'esplorazione con una strategia generica per raccogliere informazioni approssimative sull'ambiente e, in un secondo momento, utilizzare una strategia specifica legata all'obiettivo. Un ulteriore aspetto che può essere affrontato è l'implementazione della strategia proposta in un sistema multi-robot, capace di gestire compiti troppo complessi per un singolo agente.

La tesi è stata strutturata nel modo seguente.
Nel Capitolo 2 è presentato lo stato dell'arte sulle tecniche di esplorazione. Queste rappresentano la base di partenza del nostro lavoro e sono state analizzate sotto diversi punti di vista. L'accento è posto sulle strategie, i metodi utilizzati in letteratura e sulle informazioni che è stato possibile sfruttare per raggiungere il nostro scopo.

Nel Capitolo 3 è presentato in dettaglio l'ambiente di sviluppo in cui è stato implementato il nostro approccio. Sono descritte alcune scelte di progetto, come l'utilizzo di un simulatore in grado di ricreare, con grande flessibilità, un ambiente virtuale in cui eseguire test e simulazioni di agenti robotici autonomi. E' inoltre presentata l'architettura generale dell'applicazione alla base del nostro lavoro di tesi, che prende il nome dal team PoAReT. Successivamente, partendo dal lavoro svolto da questo team, sono presentate le modifiche apportate ai moduli per poter adattare il codice al nostro scopo di tesi. Le modifiche principali sono state apportate ai moduli Exploration e Path Planner, incaricati di valutare le destinazioni possibili e di calcolare i percorsi per raggiungerle. I moduli restanti non hanno subito variazioni degne di nota, rendendo così possibile il loro riutilizzo.

Nel Capitolo 4 sono presentate le fasi principali affrontate per il raggiungimento dello scopo di questa tesi. Il nostro lavoro si è concentrato sull'implementazione, all'interno del sistema PoAReT, di una variante del framework Multi-Criteria-Decision-Making (MCDM), denominata MCDM-Path, capace di valutare non solo la posizione di destinazione, ma anche il percorso necessario per raggiungerla.

Nel Capitolo 5 sono illustrate le prove sperimentali svolte e i risultati ottenuti per valutare il sistema sviluppato e verificare l'efficacia dell'utilizzo dei path come criterio per l'esplorazione.

Nel Capitolo 6, infine, sono commentati i risultati ottenuti e presentate le considerazioni finali sul lavoro svolto. Inoltre, sono presentati in dettaglio i

possibili sviluppi futuri del progetto.

Capitolo 2

Stato dell'arte

In questo capitolo viene presentato lo stato dell'arte riguardo al problema dell'esplorazione di ambienti inizialmente sconosciuti, per mezzo di robot mobili autonomi con una descrizione di alcune delle tecniche ad oggi disponibili. Nella prima sezione di questo capitolo viene presentato il problema dell'esplorazione, insieme ad alcune delle strategie proposte in letteratura per risolverlo. Nella seconda sezione viene descritta in dettaglio la strategia di Tovar et al., mentre nell'ultima sezione viene presentato il framework Multi-Criteria-Decision-Making (MCDM).

2.1 Esplorazione di ambienti

Per *esplorazione* si intende quel processo (Figura 2.1) che permette ad un agente robotico autonomo di percepire un ambiente inizialmente sconosciuto, per crearne una rappresentazione (mappa) necessaria per muoversi al suo interno. Ad un robot possono essere assegnati molteplici obiettivi che vanno dalla semplice mappatura di un spazio alla ricerca mirata di oggetti o persone. Per raggiungere questi scopi è necessario scegliere iterativamente la prossima posizione da raggiungere rispettando certi limiti di tempo e di consumo.

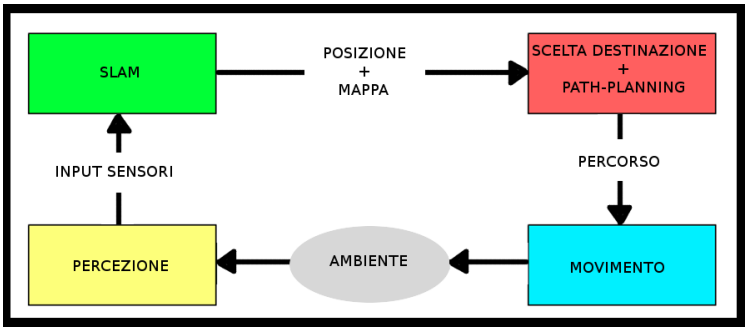

Figura 2.1: Diagramma delle fasi del processo di esplorazione.

Il tipo di ambiente in cui è necessario muoversi influenza in modo sensibile i problemi da affrontare per compiere l'esplorazione. Le tipologie principali in cui è possibile raggruppare gli spazi esplorabili sono: ambienti interni e ambienti esterni. Con ambienti interni intendiamo spazi chiusi, caratterizzati da una struttura regolare che facilita i movimenti del robot. Contrariamente ai precedenti, gli ambienti esterni presentano numerosi impedimenti alla mobilità di un robot a causa della presenza di strutture irregolari e da fenomeni esterni (per esempio: vento, pioggia, presenza di detriti, ...) che potrebbero influenzarne il movimento.

Il processo di creazione ed aggiornamento di una mappa viene comunemente chiamato Simultaneous Localization and Mapping (SLAM) [17] e consiste nell'elaborazione delle percezioni del robot al fine di ottenere la miglior rappresentazione dello spazio che lo circonda. In base alle informazioni che si ritengono più importanti (per esempio: ostacoli presenti, cammini percorribili, ...) è possibile scegliere la tipologia di mappa più adatta (Figura 2.2); tra le più comunemente adottate ci sono: mappa a griglia [17], rappresentazione tramite grafo [1] o mappa poligonale.

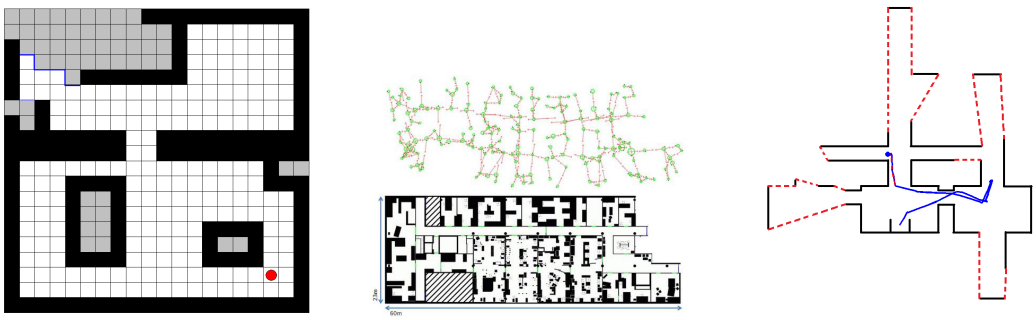

Figura 2.2: La prima immagine a sinistra è di una mappa a griglia, al centro si può osservare una mappa con grafo di supporto e sulla destra abbiamo una mappa poligonale.

Tutte queste rappresentazioni presentano vantaggi e svantaggi in termini di spazio di memoria occupato, completezza delle informazioni e complessità della loro costruzione. Le mappe a griglia rappresentano lo spazio circostante attraverso una divisione in celle che identificano l'area libera o gli eventuali ostacoli presenti al loro interno. Nell'implementazione tramite grafo, invece, le posizioni che il robot può assumere nella mappa vengono identificate tramite i nodi, mentre gli archi rappresentano i movimenti che il robot può compiere per muoversi da una posizione ad un'altra. Le mappe poligonali permettono la costruzione di una rappresentazione dell'ambiente molto più dettagliata rispetto alle due alternative presentate (a parità di dettagli una mappa a griglia richiede un consumo di memoria molto più elevato), penalizzandone però la semplicità di estrazione delle informazioni, grazie all'utilizzo di linee e segmenti per delineare gli ostacoli presenti. Dalla mappa così ottenuta è possibile identificare l'area in cui il robot può muoversi in sicurezza, definita comunemente *safe region*, separata dallo spazio sconosciuto mediante dei segmenti chiamati *frontiere* (Figura 2.3).

Per costruire in modo efficiente una mappa di un ambiente sono state sviluppate tecniche Next-Best-View (NBV), definite per scegliere la migliore posizione da raggiungere per effettuare la prossima scansione [11]. La scelta di un punto su una frontiera permette di aumentare la quantità dell'informazione aggiuntiva acquisita durante la percezione (Figura 2.3).

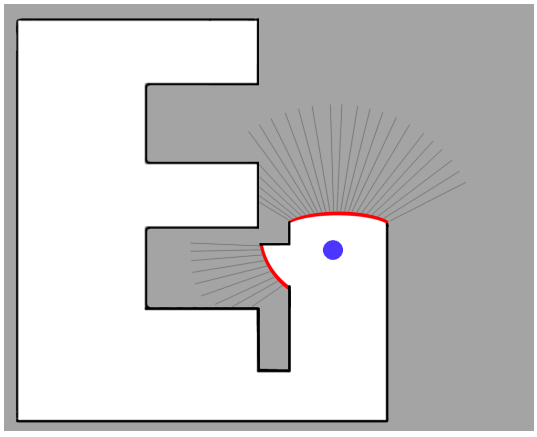

Figura 2.3: Quantità stimata dell'informazione visibile da una frontiera. In blu è indicata la posizione del robot, in rosso le frontiere [11].

Per massimizzare le prestazioni, valutate in modi diversi in base al compito che è stato assegnato al robot, è necessario che la scelta tra i possibili candidati avvenga valutandone le caratteristiche attraverso una funzione di valutazione. Tramite questa funzione, ad ogni posizione candidata viene assegnata un'utilità valutata considerando diversi criteri riferiti ad aspetti dell'ambiente, combinati in modo da identificare il punto migliore in base a dei criteri scelti. In letteratura molte di queste funzioni si basano su approcci ad hoc che non garantiscono un elevato grado di flessibilità. Alcuni esempi di questi approcci sono:

- La strategia proposta da Burgard et al. [17]:

$$u\left(p\right) = A\left(p\right) - \beta \cdot d\left(p\right) \tag{2.1.1}$$

  che valuta una posizione candidata p combinando il beneficio atteso $A(p)$ con la distanza $d\left(p\right)$ per raggiungerla. Il coefficiente β ha il compito di bilanciare il peso relativo dei benefici rispetto al costo (i ricercatori hanno dimostrato che la scelta di β ricade nell'intervallo [0.01, 50] e non provoca significative variazioni nelle prestazioni di esplorazione).

- La strategia impiegata da Amsterdam Oxford Joint Rescue Forces (AOJRF) [19]: proposta per la competizione RoboCup 2009, utilizza la seguente funzione $u\left(f\right)$ per valutare la frontiera f:

$$u\left(f\right) = \frac{A\left(f\right) \cdot P\left(f\right)}{d\left(f\right)} \tag{2.1.2}$$

  che combina la distanza $d(f)$ tra la posizione attuale del robot e la frontiera presa in considerazione, l'information gain $A(f)$ approssimata dalla lunghezza della frontiera e la probabilità di successo nella comunicazione dalla frontiera candidata $P\left(f\right)$.

- La strategia di Yamauchi [23]: seleziona la posizione migliore tra le candidate minimizzando la distanza (cioè, scegliendo sempre la posizione più vicina alla posizione attuale del robot).

Per ovviare alla limitata flessibilità degli approcci precedenti è stato proposto il framework Multi-Criteria-Decision-Making (MCDM) [8], una soluzione con solide basi matematiche e teoriche che permette la combinazione di più criteri per la valutazione delle posizioni candidate durante la fase di esplorazione.

Utilizzando un approccio NBV, dopo aver identificato la posizione da raggiungere, usando una funzione di valutazione, è necessario calcolare un percorso che permetta al robot di muoversi in sicurezza nella mappa conosciuta per raggiungere il prossimo punto dove eseguire la percezione. Questa fase, definita *path planning* [1,15] (Figura 2.4), elabora le informazioni sull'ambiente, salvate nella mappa, per generare una serie di posizioni intermedie, raggiungibili tramite semplici azioni di traslazione e rotazione facilmente eseguibili dal robot (Figura 2.5).

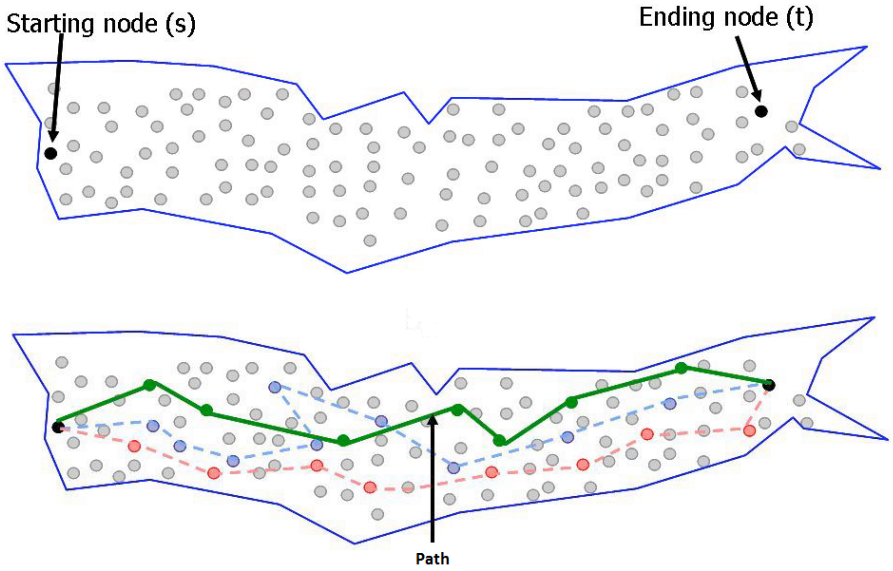

Figura 2.4: Processo di creazione di tre path per un singolo robot verso la destinazione scelta [1].

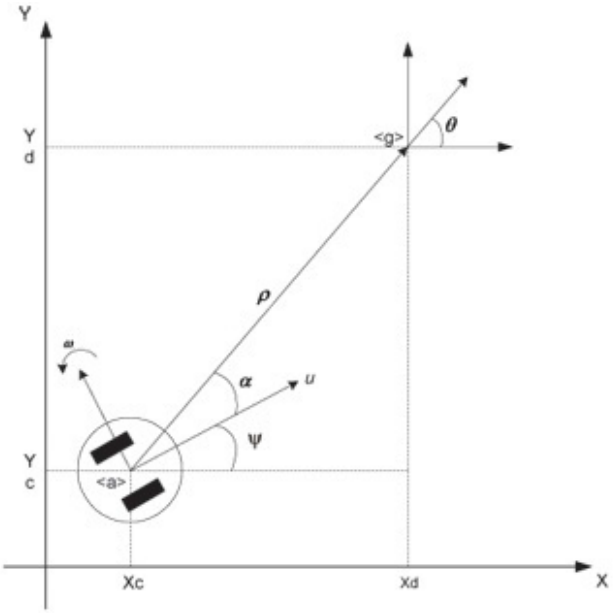

Figura 2.5: Per effettuare un movimento da $\langle a \rangle$ a $\langle g \rangle$, il robot deve compiere una rotazione di α gradi e una traslazione ρ di metri.

La quasi totalità delle strategie disponibili, oltre ad avere una scarsa flessibilità, è caratterizzata dal valutare esclusivamente il punto scelto trascurando quasi completamente il percorso necessario per raggiungerlo. Tra le eccezioni c'è la proposta di Tovar et al. [18] che considera l'informazione aggiuntiva ottenuta dal robot durante il movimento per raggiungere la destinazione, valutando così i percorsi nella loro interezza.

2.2 Strategia di esplorazione di Tovar et al.

Tra le strategie di esplorazione proposte fino ad oggi è doveroso studiare in dettaglio la soluzione implementata da Tovar et al. [18] che, come detto, si distingue dalla altre per l'attenzione posta non solo alla valutazione delle caratteristiche della destinazione, ma anche delle caratteristiche del percorso scelto per raggiungerla. Il cammino diventa così non solo un mezzo per raggiungere la destinazione, ma uno strumento che permette di incrementare l'informazione acquisita sull'ambiente. Questa strategia è realizzata attraverso una funzione di utilità che valuta i cammini verso una possibile destinazione, misurandone la qualità per mezzo di diversi criteri. Questi criteri stimano il guadagno d'informazione e la precisione della mappa ottenuta, seguendo quel particolare percorso, considerando principalmente all'accuratezza dei movimenti del ro-

bot e dall'eventuale presenza di landmark o tratti distintivi che caratterizzano l'ambiente.

Durante il movimento per seguire il percorso scelto, è possibile sfruttare i cambi di direzione o le pause per effettuare ulteriori percezioni dell'ambiente circostante. Per questo motivo nella funzione di utilità sono stati inseriti dei criteri (Figura 2.6) che permettono la stima dell'informazione aggiuntiva ricavata durante il movimento del robot.

Come detto precedentemente, al robot devono essere inviate delle azioni facilmente eseguibili e per questo motivo il percorso deve essere diviso in una serie di semplici rotazioni e traslazioni (Figura 2.5). Ognuno di questi movimenti comporta però delle imprecisioni di posizionamento che, se sommate, possono portare a dei grossi errori durante la fase di mappatura. L'utilizzo di parametri che tengono conto della frequenza di rotazioni e stop durante il movimento, permette di favorire i percorsi che garantiscono una maggiore precisione nel posizionamento dell'agente autonomo. L'accuratezza della mappatura può essere incrementata ulteriormente estraendo delle caratteristiche distintive dell'ambiente, chiamate landmark, identificabili durante la fase di percezione attraverso i sensori del robot ed utilizzabili durante il processo SLAM.

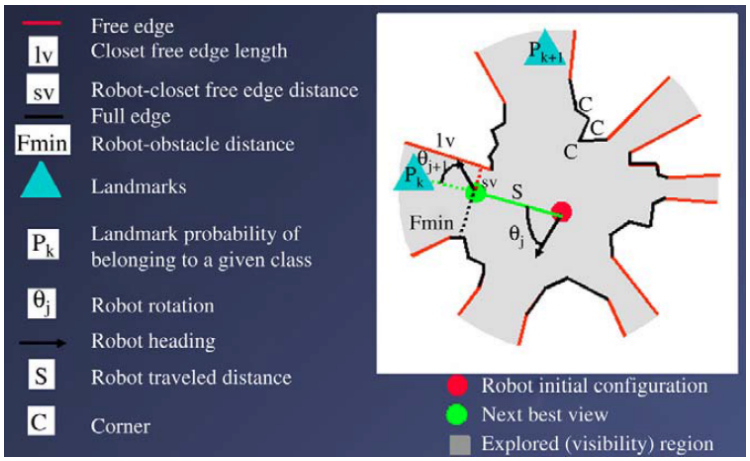

Figura 2.6: Parametri della funzione di utilità di Tovar et al. [18].

In particolare, la funzione di utilità usata da [18] per calcolare l'utilità di un path è definita dalla seguente espressione:

$$T_i = \sum_{i=1}^{m} \left(e^{(l_{v_i} - s_{v_i})} \prod_{j=1}^{q_i} \left(\frac{e^{-|\theta_j|}}{\sqrt{s_j} + 1} \right) \cdot \left(\frac{1}{n_i} \sum_{k=1}^{n_i} p_k + Ne_i \right) fmin_i\,(d_l) \right) \quad (2.2.1)$$

- i posizione in cui viene eseguita una scansione lungo il path;
- m numero totale di operazioni di rilevamento lungo il path;
- l_{v_i}lunghezza della frontiera più vicina alla posizione i;
- s_{v_i}distanza della prossima posizione i al centro della frontiera più vicina;
- j indice delle posizioni assunte dal robot proseguendo lungo il path;
- q_i numero totale di stop del robot per raggiungere la posizione i;
- θ_j cambiamento di orientamento per raggiungere la configurazione successiva j;
- s_j distanza tra la posizione j e la posizione successiva $j+1$;
- k indice di un determinato landmark;
- n_i numero di landmark all'interno della regione visibile dalla posizione i;
- p_k probabilità di identificare un landmark dalla posizione i;
- Ne_i numero di spigoli (corner) all'interno della regione visibile dalla posizione i;
- $fmin_i()$ funzione che penalizza una vicinanza eccessiva agli ostacoli;
- d_l minima distanza da un ostacolo.

Il significato di questa funzione può essere illustrato come segue:

- $e^{\left(l_{v_i} - s_{v_i}\right)}$ favorisce l'ampiezza della frontiera più vicina rimuovendo il contributo del prossimo step;
- $\prod_{j=1}^{q_i} \left(\frac{e^{-|\theta_j|}}{\sqrt{s_j}+1}\right)$ penalizza rotazioni e spostamenti eccessive tra step;
- $\left(\frac{1}{n_i}\sum_{k=1}^{n_i} p_k + Ne_i\right)$ media pesata delle somme delle probabilità di vedere spigoli e landmark;
- $fmin_i\left(d_l\right)$ penalizza un'eccessiva vicinanza agli ostacoli.

E' doveroso precisare che nel documento in cui è presentata questa funzione di valutazione, non viene specificato il metodo utilizzato per la generazione dei percorsi.

Per la valutazione sperimentale della qualità della strategia proposta da Tovar et al. sono stati effettuati dei confronti rispetto ad altre proposte, considerando la lunghezza dei path percorsi, il numero delle rotazioni e degli stop che il robot ha compiuto ed il numero delle percezioni effettuate. Questi confronti sperimentali sono stati eseguiti in ambientazioni indoor utilizzando sia degli agenti autonomi reali che effettuando simulazioni. Dai risultati ottenuti la strategia si è dimostrata molto efficace nella valutazione di interi percorsi senza incorrere in un costo computazionale troppo elevato.

2.3 MCDM

L'importanza della funzione di valutazione nell'esplorazione ha portato numerosi autori alla creazione di versioni ad hoc che si adattassero al meglio alle proprie necessità e alle caratteristiche dell'ambiente in esame. Tutto questo ha causato la nascita di molte versioni poco flessibili di funzioni di utilità adatte esclusivamente allo scopo per il quale sono state create [2, 18]. Per questo motivo alcuni ricercatori hanno posto la propria attenzione alla creazione di soluzioni più generali, in grado di adattarsi alle diverse necessità. Un esempio è il framework Multi-Criteria-Decision-Making (MCDM) [8], caratterizzato da un'elevata flessibilità ed implementato su solide basi matematiche e teoriche. La funzione di valutazione è ottenuta tramite la combinazione di due elementi fondamentali: i criteri (per esempio: batteria consumata, distanza percorsa, guadagno d'informazione sull'ambiente,...), che permettono di valutare le caratteristiche ritenute più importanti di una possibile configurazione del robot, e i pesi, i quali permettono la combinazione dei diversi criteri mettendo in risalto quelli ritenuti più importanti.

La funzione di utilità risultante è la seguente:

$$f(p) = \sum_{j=1}^{n} \left(\left(u_j(p) - u_{j-1}(p) \right) \cdot \mu(A_j) \right) \tag{2.3.1}$$

- j indice del criterio, organizzati in ordine di utilità crescente;
- n numero totale di criteri presi in considerazione;

- $u_j\,(p)$ funzione di valutazione normalizzata del criterio j-esimo applicata alla posizione p;
- $\mu\,()$ funzione di valutazione normalizzata di un insieme di criteri;
- A_j insieme dei criteri con utilità maggiore o uguale al criterio j.

Nell'equazione 2.3.1 si utilizza l'integrale discreto di Choquet [12], un operatore di aggregazione che permette la combinazione di un numero qualsiasi di criteri. Per garantire che il risultato finale sia coerente, utilizzando una funzione di valutazione che restituisce sempre un valore positivo, è necessario stabilire un ordinamento crescente dei criteri. Questo implica che il valore del criterio j-esimo deve essere maggiore del valore del criterio $(j-1)$-esimo.

2.3.1 Caratteristiche dei pesi

I pesi relativi ai criteri utilizzati devono essere sottoposti ai seguenti vincoli:

Positività

$$\forall i \ \in \mathcal{P}\,(Criteri)\,, \quad \omega_i \geq 0 \tag{2.3.2}$$

dove:

- $\mathcal{P}\,(Criteri)$ è l'insieme di tutti i sotto-insiemi di criteri;
- ω_i è il peso del criterio i-esimo.

Somma a uno dei pesi individuali:

$$\sum_{i=1}^{n} \omega_{c_i} = 1 \tag{2.3.3}$$

dove ricordiamo che n è il numero di criteri.

Monotonia: dato un insieme di criteri $C = \{c_1, c_2, \ldots, c_n\}$, il peso di una loro combinazione $C = \langle c_1, c_2, ..., c_j \rangle$, con $C_1 \subseteq C$, deve essere sempre maggiore o uguale al peso di una combinazione costituita dagli elementi di un insieme $C_2 \subseteq C_1$. Quindi:

$$\omega_{c_j} \leq \omega_{c_i} \quad \forall C_i, C_j \mid C_j \subseteq C_i \tag{2.3.4}$$

Si noti che, per i vincoli espressi in precedenza, tutti i pesi appartengono all'intervallo $[0, 1]$.

Le relazioni che influenzano i pesi dei criteri possono essere formalizzate come segue:

Sinergia: se due criteri c_1 e c_2 sono sinergici, allora il peso della loro combinazione dovrà essere maggiore della somma dei singoli pesi:

$$\omega_{\{c_1,c_2\}} > \omega_{c_1} + \omega_{c_2} \tag{2.3.5}$$

Ridondanza: se due criteri c_1 e c_2 sono ridondanti, allora il peso della loro combinazione dovrà essere minore della somma dei singoli pesi:

$$\omega_{\{c_1,c_2\}} < \omega_{c_1} + \omega_{c_2} \tag{2.3.6}$$

Indipendenza: se due criteri c_1 e c_2 sono indipendenti, allora il peso della loro combinazione dovrà essere pari alla somma dei singoli pesi:

$$\omega_{\{c_1,c_2\}} = \omega_{c_1} + \omega_{c_2} \tag{2.3.7}$$

2.3.2 Proprietà di MCDM

Nel caso in cui i vincoli precedenti siano soddisfatti, la funzione di utilità definita nell'equazione 2.3.1 assicura il mantenimento delle seguenti proprietà matematiche:

Sia u_p il vettore delle utilità associate al candidato p, $u_p = (u_1(p), u_2(p), \ldots, u_n(p))$
Monotonia in ogni argomento: per ogni $u_p, u_p^{'} \in I^n$:
dove:

- $I = [0, 1]$;
- N: insieme dei criteri;

$$se\ \forall i \in N, u_i(p) \leq u_i^{'}(p) \Longrightarrow f(u_p) \leq f\left(u_p^{'}\right) \tag{2.3.8}$$

$$se\ \forall i \in N, u_i(p) < u_i^{'}(p) \Longrightarrow f(u_p) < f\left(u_p^{'}\right) \tag{2.3.9}$$

Questa proprietà assicura che, tra un insieme di punti candidati, la funzione di valutazione $f()$ ne selezionerà uno tra quelli Pareto-dominanti[1].

Stabilità nelle trasformazioni lineari: per ogni $u_p \in I^n$ e per ogni $r, s \in \mathbb{R}$ con $r > 0$ tale che, per ogni $i \in N$, $ru_i(p) + s \in I$, si verifica che:

$$f(ru_1(p) + s, ru_2(p) + s, \ldots, ru_n(p) + s) = rf(u_1(p), u_2(p), \ldots, u_n(p)) + s \tag{2.3.10}$$

Questo stabilisce che le valutazioni restituite dalla funzione sono indipendenti dalla scala. Si può quindi assumere, senza perdere generalità, che la funzione $f()$ restituisca un output appartenente all'intervallo $I = [0, 1]$.

Continuità: Dato un numero di criteri N, la funzione di aggregazione $f()$ è continua nell'insieme delle valutazioni I^n. Questo assicura che a lievi variazioni degli argomenti non vi siano grandi variazioni nel risultato della funzione di aggregazione.

Idem-potenza: se per un punto candidato p, tutte le valutazioni $u_i(p) = u \in I$ allora:

$$f(u_1(p), u_2(p), \ldots, u_n(p)) = f(u, u, \ldots, u) = u \tag{2.3.11}$$

Questa proprietà assicura una consistenza nelle valutazioni: se tutti i criteri restituiscono lo stesso valore, allora anche l'output della funzione di valutazione avrà il medesimo risultato. Tale proprietà garantisce che un criterio non abbia maggiore importanza strutturale rispetto ad un altro.

[1]Un punto Pareto-dominante è un punto per il quale non esistono alternative che siano migliori contemporaneamente per tutti i criteri considerati.

Capitolo 3

Descrizione ambiente di sviluppo

In questo capitolo vengono presentati gli strumenti utilizzati per lo sviluppo del nostro lavoro di tesi; nello specifico vengono presentati gli aspetti caratteristici dell'applicazione USARSim, utilizzata per creare un ambiente virtuale nel quale è stato possibile eseguire le simulazioni per valutare le prestazioni del codice prodotto. La seconda parte del capitolo è destinata alla descrizione del software PoAReT, base di partenza dalla quale è partito il nostro lavoro di tesi. Successivamente vengono presentati, in modo più dettagliato, i moduli da noi modificati per il raggiungimento dello scopo di questa tesi.

3.1 USARSim

Il sistema proposto in questa tesi si riferisce principalmente ad un contesto di Urban Search And Rescue (USAR). In questo tipo di situazioni, generate in seguito a disastri naturali o incidenti, si rende necessario cercare eventuali vittime o superstiti all'interno dell'ambiente colpito. In questi contesti, che possono comprendere terremoti, incendi e crolli di abitazioni, l'utilizzo di agenti robotici autonomi o semi-autonomi può portare a numerosi benefici eliminando o limitando i rischi legati ad un intervento diretto dei soccorritori salvaguardando la loro vita. Nonostante questa promettente premessa, i sistemi ad oggi disponibili soffrono di svariate problematiche. In una situazione di emergenza un robot completamente autonomo difficilmente può godere della piena fiducia delle persone coinvolte, portando le possibili vittime a rifiutarne l'aiuto. Problemi possono sorgere anche dal lato tecnico, l'inaffidabilità dei componenti rende molto difficile l'implementazione di un sistema di riconoscimento com-

pletamente automatico in grado di riconoscere le vittime; questo porta alla necessità di un intervento umano per la conferma delle percezioni [6].

Negli scenari reali, la conformazione del terreno può complicare ulteriormente le problematiche affrontate dal robot aumentando la complessità del modulo dedicato alla locomozione e all'esplorazione. Per questo motivo è stato proposto USARSim [9], un simulatore in grado di ricreare un ambiente virtuale in cui eseguire test e simulazioni di agenti autonomi, ottenendo dei risultati paragonabili a quelli conseguiti dalle controparti reali dei robot utilizzati .

Il simulatore USARSim[1] è costruito sul motore grafico UDK[2] in grado di ricreare fedelmente sia l'aspetto che la fisica di ambientazioni reali (Figura 3.1). Questa applicazione offre una grande flessibilità nella creazione di mappe (Figura 3.2) e di robot.

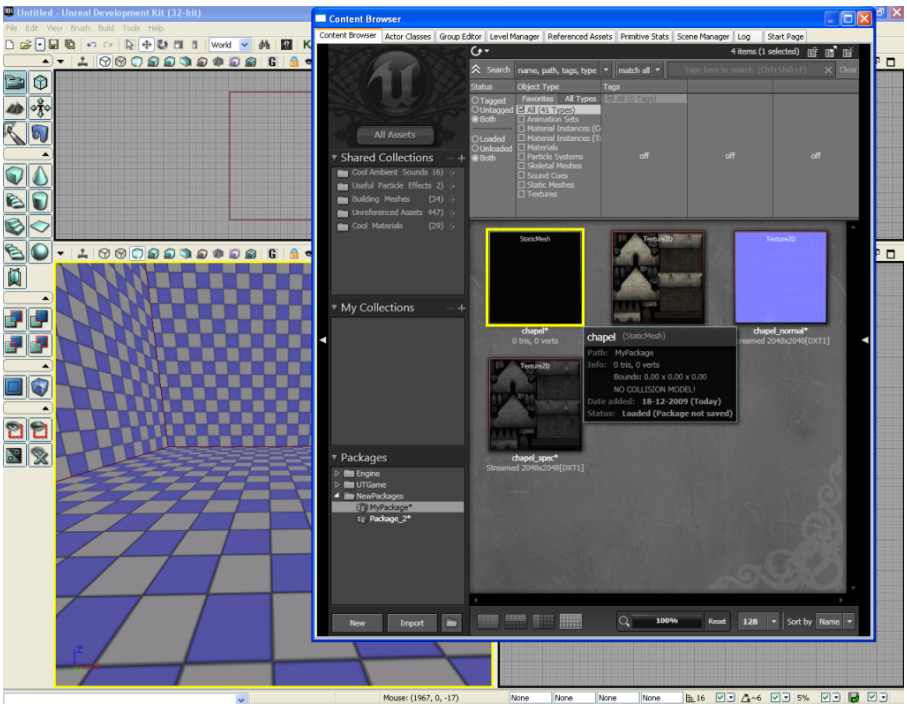

Figura 3.1: Esempio del motore grafico UDK.

[1]http://sourceforge.net/p/usarsim/code
[2]https://www.unrealengine.com

Figura 3.2: Esempio di un ambiente generato con USARSim.

L'utilizzo di un simulatore permette agli sviluppatori di concentrarsi sugli aspetti di alto livello della propria applicazione, tralasciando molti degli aspetti di basso livello che renderebbero troppo complicata l'implementazione e l'esecuzione dei test sperimentali nelle fasi iniziali dello sviluppo.

Un ulteriore vantaggio dell'utilizzo di un simulatore è la ripetibilità degli esperimenti, USARSim permette infatti a sviluppatori di diverse parti del mondo di eseguire i propri test nelle stesse condizioni, garantendo un migliore confronto dei risultati [5].

3.2 PoAReT

L'applicazione alla base del nostro lavoro di tesi prende il nome dal "Politecnico di Milano Autonomous Robotic Rescue Team"[3] (PoAReT), partecipante alla competizione RoboCup2012 Rescue Virtual Robots (Figura 3.3) con l'obiettivo di controllare un sistema multi-robot per trovare il maggior numero di vittime in un ambiente simulato e ignoto a priori.

[3]http://home.dei.polimi.it/amigoni/research/PoAReT.html

Figura 3.3: Logo della competizione RoboCup2012, tenutasi a Città del Messico.

Questa applicazione fa uso del simulatore USARSim descritto in precedenza (Sezione 3.1). Lo scopo del team è stato quello di sviluppare un controllore in grado di interfacciarsi al simulatore per poter inviare comandi, contenenti le istruzioni per il robot, agli agenti impegnati nella competizione.

Grazie all'elevato grado di autonomia del sistema PoAReT, i robot del sistema sono riusciti ad esplorare la maggior parte della mappa nel tempo a disposizione, identificando il maggior numero di vittime. Questo è stato il grande vantaggio rispetto ai team concorrenti che hanno fatto affidamento esclusivo sui comandi inviati da un singolo operatore umano a un singolo robot, lasciando sostanzialmente inattivi tutti gli altri agenti. Tale capacità di esplorazione autonoma ha portato il team PoAReT alla vittoria della competizione internazionale.

3.2.1 Architettura generale

Il sistema (Figura 3.4) che controlla la squadra di robot del team PoAReT è composto principalmente da due componenti: la stazione di comunicazione e il sistema di controllo di un singolo robot. La stazione, chiamata anche *"Base Station"*, costituisce l'interfaccia grafica del sistema e si occupa della gestione delle comunicazioni tra i robot. Ogni singolo agente viene eseguito in dei processi dedicati, generati dall'applicazione chiamata *"Poaret"*, responsabile del controllo del robot.

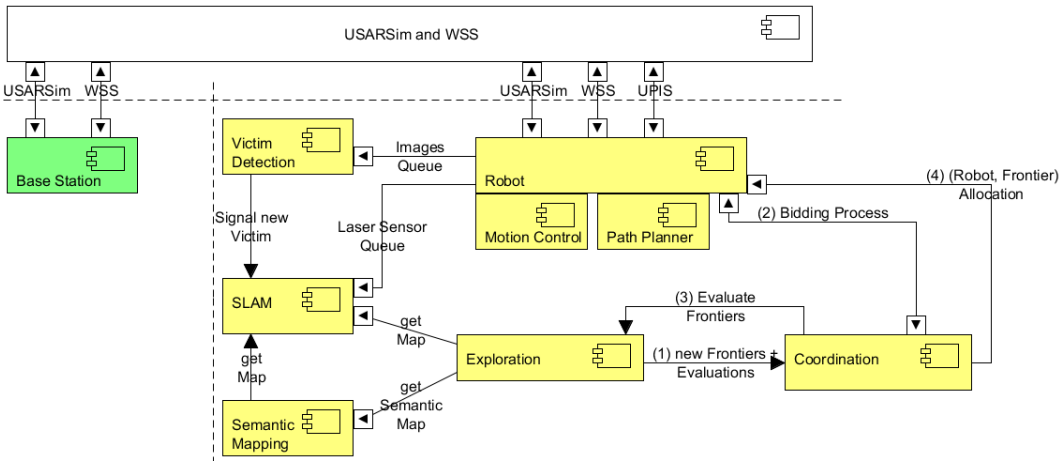

Figura 3.4: Architettura del sistema

Dalla figura 3.4 è possibile notare che il sistema interagisce, tramite opportuni socket, con i seguenti componenti esterni:

- **USARSim:** il simulatore in grado di ricreare un ambiente virtuale in cui eseguire test e simulazioni di agenti autonomi (Sezione 3.1);

- **Wireless Simulation Server (WSS):** il processo responsabile della creazione di una rete wireless simulata all'interno dell'ambiente virtuale.

Il robot preso come riferimento dal team PoAReT è il modello Pioneer 3AT che possiede quattro ruote motrici ed è equipaggiato con due sensori laser "SICK" (ciascuno con un campo visivo di 180°, raggio di 20m e risoluzione di 1°, che, montati alla stessa altezza, permettono di avere un campo visivo di 360°), due cinture composte da 5 sonar, una anteriore ed una posteriore, un odometro ed una telecamera.

L'architettura è composta da diversi moduli indipendenti, eseguiti su thread dedicati, coordinati dal componente *Robot* incaricato di inoltrare i messaggi interni al sistema o verso il simulatore. I moduli più rilevanti per il nostro lavoro di tesi sono i seguenti:

- **Motion Controller:** si occupa di raccogliere i messaggi provenienti dagli altri moduli, convertendoli in segnali per gli attuatori che permettono il movimento del robot. Smista, inoltre, le informazioni sul mondo esterno provenienti dai sensori, indirizzandole verso i moduli che ne fanno richiesta;

- **Simultaneous Localization And Mapping (SLAM):** interpreta gli input sensoriali provenienti dal sensore laser per creare una rappresentazione (mappa) dell'ambiente;

- **Exploration:** attua la politica di valutazione della strategia scelta per decidere il prossimo punto da raggiungere;

- **Path Planner:** genera il percorso necessario per raggiungere la destinazione scelta.

3.2.2 Motion Controller

Come accennato in precedenza, il modulo RobotController si occupa della gestione delle informazioni in ingresso ed in uscita. I segnali inviati verso l'esterno controllano gli attuatori che si occupano del movimento del robot; questi comandi vengono generati elaborando le informazioni ricevute dal Path Planner (Sezione 3.2.5), convertendole in una serie di semplici rotazioni e traslazioni da eseguire. Questo modulo, per quanto riguarda i segnali in ingresso, riceve le informazioni provenienti dai sensori ed agisce di conseguenza. Un segnale generato dal sonar può indicare la presenza di un ostacolo e richiedere l'arresto del robot, mentre le informazioni provenienti dal laser vengono reindirizzate verso il modulo SLAM (Sezione 3.2.3), dove sono utilizzate per creare una rappresentazione dell'ambiente circostante.

3.2.3 SLAM

Questo modulo, ricevuti gli input provenienti dal laser e dall'odometro, ha il compito di ricreare una mappa il più fedele possibile all'ambiente circostante in cui il robot agisce.

La tipologia scelta per rappresentare lo spazio è una mappa poligonale (Figura 2.2); l'utilizzo di segmenti per identificare ostacoli e frontiere è infatti adatto alla rappresentazione di ambienti interni con in più il vantaggio, rispetto ad approcci alternativi, di un impatto minore sul consumo della memoria e sulla quantità di informazioni scambiata.

3.2.4 Exploration

Il modulo Exploration, dopo aver considerato lo stato attuale del robot e la mappa nota fino ad un certo istante, ha il compito di effettuare la valutazione delle frontiere alla ricerca del miglior punto da raggiungere da cui eseguire la successiva scansione dell'ambiente. Diversamente dagli altri moduli, che si attivano una volta ricevuti degli input, questo componente si attiva alla scadenza di un timer che ne regola l'esecuzione e che, in caso di inattività dell'agente robotico, riduce i tempi di attesa e permette di percepire con maggiore reattività eventuali momenti di stallo.

3.2.5 Path Planner

Scelta una destinazione attraverso il modulo Exploration (Sezione 3.2.4), è necessario calcolare il percorso più adatto per raggiungerla. Questa fase, definita Path Planning [14], è effettuata attraverso una versione modificata dell'algoritmo A^* (Appendice B), chiamata *Hybrid* A^*. Questo algoritmo discretizza lo spazio nell'intorno del robot, trasformando in celle solo le porzioni di area interessanti, al fine di trovare le posizioni intermedie raggiungibili con semplici rotazione e traslazioni. Per aumentare l'efficienza è stato scelto di sacrificare l'ottimalità dell'algoritmo, accettando anche delle soluzioni sub-ottime.

3.2.6 Interfaccia utente

Questo componente si occupa di fornire un'interfaccia (Figura 3.5) agli operatori che permette la gestione ed il controllo del robot. La comunicazione tra i robot e la *Base Station* avviene tramite il processo WSS, che raccoglie ed inoltra i messaggi da e per il robot, simulando una rete wireless.

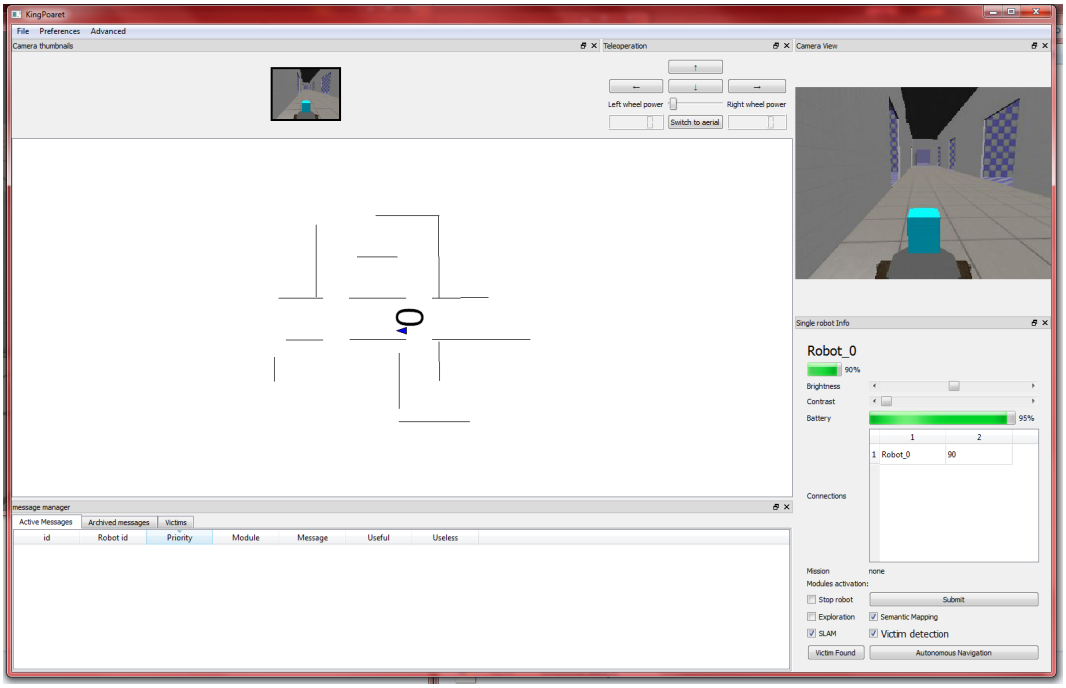

Figura 3.5: Interfaccia grafica che mostra la mappa costruita dal robot e i controlli per il movimento tramite operatore.

3.2.7 Modifiche apportate

Partendo dal lavoro svolto dal team PoAReT, abbiamo apportato delle modifiche al codice esistente per adattarlo al nostro scopo. Le modifiche principali sono state apportate ai moduli Exploration e Path Planner, incaricati di valutare le destinazioni possibili e di calcolare i percorsi per raggiungerle. I moduli restati non hanno subito variazioni degne di nota, rendendo così possibile il loro riutilizzo.

3.2.7.1 Modulo Exploration

Diversamente dal codice esistente, il nostro lavoro necessita di valutare non solo le destinazioni possibili, ma anche i cammini per raggiungerle. Per questo motivo si è reso necessario anticipare la fase di path planning rispetto a quella della valutazione. Un'ulteriore aspetto caratterizzante il nostro lavoro di tesi è l'utilizzo esclusivo di un singolo robot, questo ha permesso di semplificare la fase di assegnazione delle destinazioni, eliminando così le aste utilizzate per tale scopo (la struttura a moduli dell'applicazione PoAReT permette, comunque, di reintrodurre con semplicità le aste nel caso ce ne fosse bisogno). Queste, infatti, si rendevano necessarie nei casi multi-robot per gestire l'assegnazione

delle frontiere da raggiungere. La *Base Station* era incaricata di raccogliere le valutazioni attribuite dai robot alle frontiere raggiungibili, per poi assegnare la destinazione più adatta ad ogni agente, evitando così possibili conflitti.

3.2.7.2 Modulo Path Planner

In questo modulo è stato completamente riscritto l'algoritmo incaricato della creazione dei path. Per creare cammini che variassero in modo significativo l'uno dall'altro, si è scelto di campionare uniformemente l'ambiente conosciuto, estraendo un grafo delle posizioni raggiungibili dal robot. Questo processo, denominato *Probabilistic RoadMaps* (PRM), è seguito dalla selezione dei percorsi più promettenti attraverso $AlphA^*$ [16], un algoritmo di ricerca con il quale estrarre, modificando un parametro α, un insieme di cammini (sub)ottimi, rispetto alla lunghezza del path, verso la destinazione scelta.

Nel successivo capitolo sono state presentate, in maggior dettaglio, le modifiche apportate a questi moduli.

Capitolo 4

Implementazione di MCDM-Path

In questo capitolo vengono presentate le fasi principali affrontate per il raggiungimento dello scopo di questa tesi. Il nostro lavoro si è concentrato sull'implementazione, all'interno del sistema PoAReT, di una variante MCDM, chiamata in seguito *MCDM-Path*, capace di valutare non solo la posizione di destinazione, ma anche il percorso per raggiungerla.

L'idea generale alla base del nostro lavoro di tesi (Appendice A) è quella di, partendo dalla rappresentazione parziale dell'ambiente, generare un insieme di path verso delle destinazioni scelte sulle frontiere, che verranno valutati per poterne determinare il migliore. Il processo di creazione dei cammini è preceduto da una serie di azioni, ripetute in modo ciclico ed eseguite durante l'esplorazione dell'ambiente, facenti parte dell'algoritmo denominato PRM [13] (Algoritmo 4.1).

Nelle sezioni seguenti verranno descritte in dettaglio queste azioni, che consistono principalmente nel campionamento dello spazio libero della mappa (Sezione 4.1), per ottenere un insieme di posizioni raggiungibili dal robot, nella creazione di un grafo (Sezione 4.2), generato unendo questi posizioni, e nell'estrazione di un insieme di cammini (Sezione 4.3) percorribili dal robot.

Algoritmo 4.1 Algoritmo PRM per il campionamento della mappa e la successiva creazione del grafo.

```
void PRM(Map map){
  for(int i=0;i<pointNumber;i++){
    //Creazione nuovo punto
    point = newRandomPoint(map);
    //Controllo validità punto
    if(visibilityCheck(point,map)){
      //Aggiornamento grafo
      graph.addNode(point,map);
    }
  }
  //Aggiornamento punti frontiere
  if(pointNumberFrontier!=0){
    foreach(Frontier frontier, frontiersRemoved){
      //Cancellazione punti frontiere rimosse in seguito
      //ad una nuova percezione
      graph.deleteNodes(&frontier);
    }
    //Creazione punti nell'intorno delle frontiere
    //aggiunte in seguito ad una nuova scansione
    foreach(Frontier frontier, frontiersAdded){
      for(int i=0;i<pointNumberFrontier;i++){
        point = newRandomPointFrontier(frontier, map);
        //Controllo validità punto
        if(visibilityCheckFrontier(point,frontier,map)){
          //Aggiornamento grafo
          graph.addNode(point,map,&frontier);
        }
      }
    }
  }
}
```

4.1 Campionamento dello spazio libero

Per poter generare dei percorsi validi, che il robot possa seguire per raggiungere una possibile destinazione, è necessario estrarre dalla mappa una serie di posizioni appartenenti alla regione dell'ambiente libera da ostacoli.

Come detto in precedenza la rappresentazione dell'ambiente consiste in una mappa poligonale, composta da segmenti che identificano ostacoli e frontiere, costruita in modo incrementale con le informazioni ottenute ad ogni nuova percezione del robot.

Per generare l'insieme delle posizioni valide, si è scelto di implementare un algoritmo iterativo che, ad ogni movimento del robot, campionasse con densità uniforme (Figura 4.1) la nuova area libera scoperta. Il campionamento avviene attraverso una funzione dedicata, che genera un numero fissato di punti casuali nell'intorno del robot con densità uniforme. Nell'intorno delle frontiere viene applicato un campionamento aggiuntivo, per intensificare la presenza di punti nelle aree di maggiore interesse (Algoritmo 4.2). Da questo insieme di punti vengono eliminati quelli ritenuti non raggiungibili (Algoritmo 4.3), esterni alla mappa visibile o posizionati all'interno di un ostacolo, riducendo così l'elenco alle sole configurazioni valide.

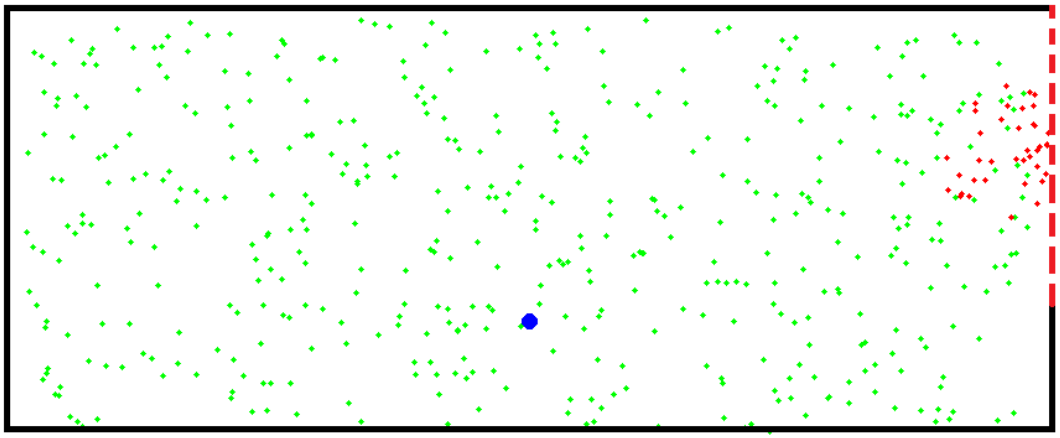

Figura 4.1: Campionamento con densità uniforme (punti verdi) sulla mappa conosciuta, con campionamento aggiuntivo nell'intorno delle frontiere (punti rossi). Il punto di colore blu identifica la posizione del robot. Le stesse convenzioni sono usate per le altre figure del capitolo.

Algoritmo 4.2 Algoritmo per la generazione di un punto casuale.

```
Point newRandomPoint(Map map){
  //Posizione Robot
  robotPose = map.lastRobotPose();
  //Generazione distanza casuale attraverso la funzione rand()-->[0,1]
  double randomDistance = rand();
  //L'utilizzo della radice della distanza permette di
  //ottenere un campionamento uniforme
  //maxDistance: parametro estratto dal file di configurazione
  double distance = maxDistance*sqrt(randomDistance);
  //Generazione angolo casuale attraverso la funzione rand()-->[0,1]
  double randomAngle = rand();
  double angle = randomAngle*2*PI;
  //Calcolo coordinate punto
  double x= robotPose->x()+distance*cos(angle);
  double y= robotPose->y()+distance*sin(angle);
  return Point(x,y);
}
```

La funzione di seguito riportata mostra il controllo della validità di un punto generato in modo casuale. Tale punto deve fare parte solo della nuova area percepita dal robot senza ricadere negli ostacoli o nei muri. Per fare questo la funzione di controllo deve tener presente delle frontiere, che devono essere aggiornate ad ogni percezione dell'agente. Posso infatti verificarsi due casi:

- Il robot non ha oltrepassato una frontiera rimossa (Figura 4.2) (cioè una frontiera presente nella versione percezione $i-1$, poi cancellata in seguito al movimento del robot). In questo caso i nuovi punti saranno validi esclusivamente se il segmento che li collega alla posizione del robot interseca la frontiera in questione;

- Il robot ha oltrepassato una frontiera rimossa (Figura 4.3). In questo caso i nuovi punti saranno validi esclusivamente se il segmento che li collega alla posizione del robot non interseca la frontiera in questione.

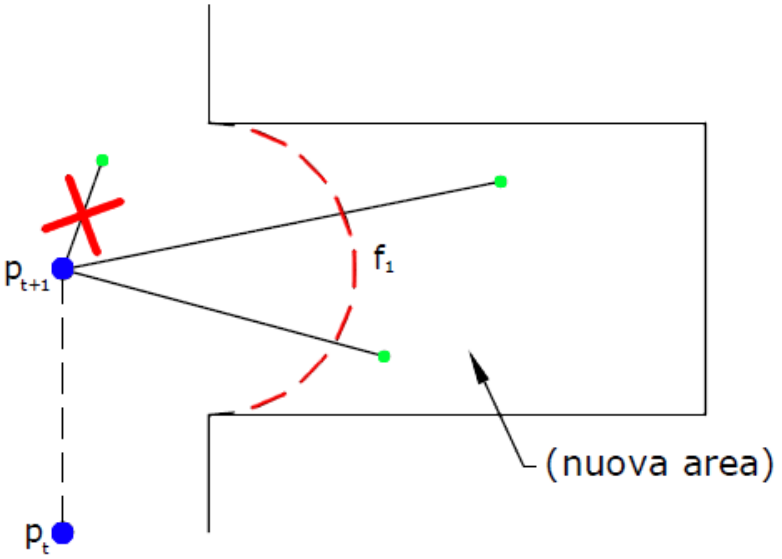

Figura 4.2: Controllo validità punto nel caso di una frontiera non oltrepassata. Il robot durante il movimento da p_t a p_{t+1} non attraversa la frontiera f_1 rimossa nell'ultimo aggiornamento della mappa.

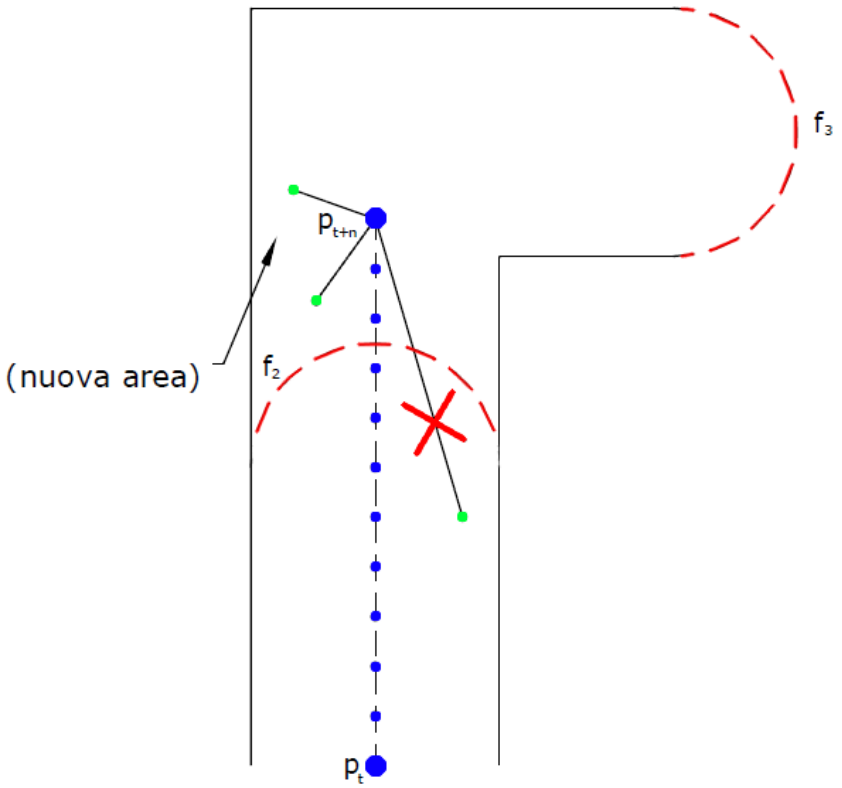

Figura 4.3: Controllo validità punto nel caso di una frontiera oltrepassata. Il robot durante il movimento da p_t a p_{t+n} attraversa la frontiera f_2 rimossa nell'ultimo aggiornamento della mappa.

Algoritmo 4.3 Algoritmo che controlla la validità di un punto casuale.

```
bool visibilityCheck (Point point, Map map){
  //Posizione Robot
  currentPose = map.lastRobotPose();
  movementLine = LineSegment(currentPose, previousPose);
  visibilityLine = LineSegment(currentPose, point);
  //Iterazione su ogni frontiera eliminata
  foreach(Frontier frontier, frontiersRemoved){
    if(frontier.intersects(movementLine)){
      //Frontiera oltrepassata
      if(!(frontier.intersects(visibilityLine))){
        //Punto oltre la frontiera
        if((map.isReachable(point,currentPose)==true){
          //Punto valido
          return true;
        }
      }
    }
    else{
      //Frontiera non oltrepassata
      if((frontier.intersects(visibilityLine))){
        //Punto oltre la frontiera
        if((map.isReachable(point,currentPose)==true
        //Punto valido
          return true;
        }
      }
    }
  }
  //Punto non valido
  return false;
}
```

4.2 Costruzione del grafo

I punti ottenuti in precedenza vengono utilizzati per creare un grafo, inizialmente vuoto, aggiungendo, in seguito, tutti gli archi (Figura 4.4) che non collidono con ostacoli e di lunghezza inferiore ad una soglia, personalizzabile per rendere il sistema più flessibile e compatibile con il maggior numero di mappe (Algoritmo 4.4). Questa struttura, aggiornata ad ogni iterazione dell'algoritmo PRM, rappresenta un sotto-insieme dei possibili cammini che il robot può percorrere in sicurezza.

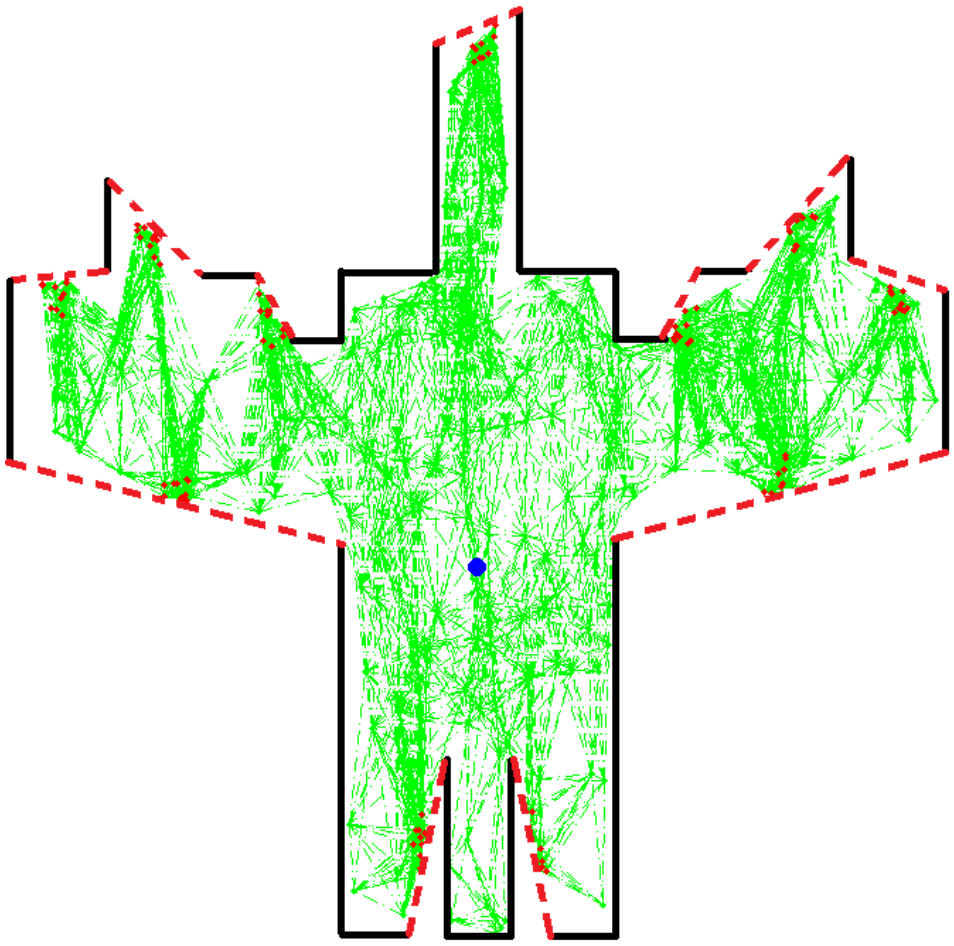

Figura 4.4: Grafo risultante dal campionamento dello spazio libero e dalla creazione degli archi.

Algoritmo 4.4 Algoritmo che crea tutti gli archi validi tra tutti i nodi del grafo.

```
void createEdges(Map map){
    //Creazione dell'insieme dei nodi come unione
    //dei nodi generati nell'intorno del robot e dei
    //nodi generati nell'intorno delle frontiere
    nodesList = nodes+frontierNodes;
    int size = nodesList.size();
    //Iterazione su tutti i nodi
    for(int i=0; i<size; i++){
        for(int j=i+1; j<size; j++){
            node1 = nodesList.at(i);
            node2 = nodesList.at(j);
            //Controllo lunghezza arco
            if(node1->distance(node2)<edgeThreshold){
            //Controllo validità arco
                if(map.isReachable(node1,node2)){
                    edge = new GraphEdge(node1,node2);
                    //Aggiornamento lista archi
                    edgesList.append(edge);
                }
            }
        }
    }
}
```

4.3 Estrazione dei percorsi

Dal grafo ottenuto in precedenza è necessario estrarre un insieme di cammini verso le destinazioni scelte. In questa fase il nostro interesse non ricade sulla scelta del percorso migliore in assoluto, ma su un sotto-insieme di path possibili, cercando di ottenere cammini con caratteristiche abbastanza diverse da rendere interessante la loro valutazione (Figura 4.5). Per fare questo si è scelto di utilizzare una versione modificata dell'algoritmo A^* (Appendice B), descritta nella prossima sezione, caratterizzata da un parametro α che permette di estrarre path diversi da un grafo (Algoritmo 4.5), sacrificando però l'ottimalità della soluzione.

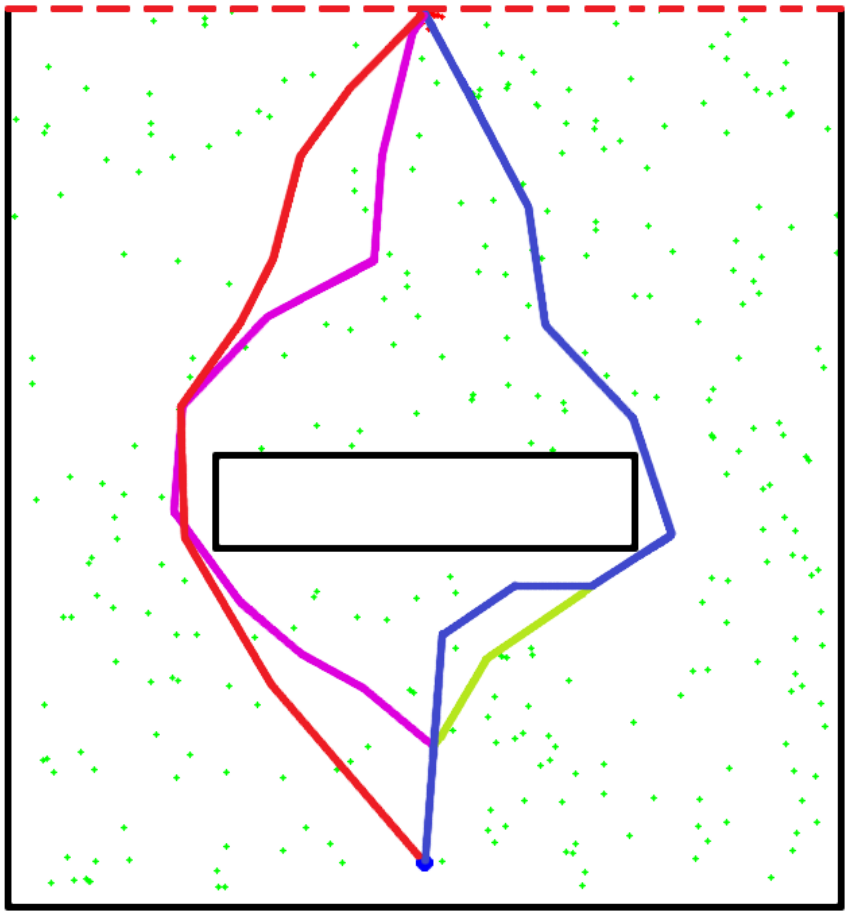

Figura 4.5: Estrazione di alcuni dei percorsi possibili dalla posizione del robot al punto di destinazione sulla frontiera.

Algoritmo 4.5 Algoritmo che estrae gli n path migliori verso una destinazione.

```
List<Path> getPaths(Point destination){
    //Iterazione sul numero di path richiesti
    for(int alpha=1; alpha<=pathNumber;alpha++){
        //Esecuzione algoritmo AlphA*
        path = aStar->getPath(alpha, destination);
        if(!path.isEmpty()){
            pathList.append(path);
        }
    }
    return pathList;
}
```

4.3.1 Algoritmo AlphA*

Questo algoritmo (4.6) [16] utilizza una versione modificata dell'algoritmo A^* (Appendice B) per ricavare più di un path (sub)ottimo dal grafo ottenuto in precedenza (Sezione 4.2). Ad ogni iterazione viene scelto il nodo, tra quelli appartenenti al grafo e non ancora scelti, con il minor valore di $f(x)$ (Equazione 4.3.1).

$$f(x) = g(x) + \alpha \cdot h(x) \tag{4.3.1}$$

dove:

- $g(x)$: costo per raggiungere il nodo x partendo dall'origine (cioè dalla posizione attuale del robot), calcolato come somma delle lunghezze degli archi attraversati;
- $h(x)$: euristica che stima il costo per raggiungere la destinazione partendo dal nodo x;
- α: coefficiente che permette di generare percorsi con un tempo computazionale minore, al costo di introdurre una possibile sub-ottimalità nella soluzione.

Partendo dal nodo che rappresenta la posizione del robot, è possibile così ricavare i percorsi verso la destinazione scelta. Per garantire il raggiungimento

di una soluzione, è necessario che i costi assegnati ad ogni arco non siano negativi. Durante l'esecuzione dell'algoritmo, la variazione del parametro α dovrebbe permette, in base alle prove sperimentali effettuate, di generare un numero soddisfacente di path diversi tra loro.

Algoritmo 4.6 Algoritmo *AlphA** per la generazione di un cammino.

```
PRMPath getPath(double alpha, Point destination){
  //Scelta nodo più vicino alla destinazione
  goalNode = graph->nearestToDestination(destination);
  start->setHValue(start->distance(destination));
  //Creazione nodo destinazione
  goal->setHValue(0);
  goal->setParent(NULL);
  goal->setNode(goalNode);
  //Inizializzazione openSet e ClosedSet
  openSet->append(start);
  closedSet->clear();
  while(!openSet->isEmpty()){
    current = chooseNextNode(alpha);
    if(current==goal){
      //Destinazione raggiunta
      path = calculatePath(current);
      return path;
    }
    openSet->remove(current);
    closedSet->append(current);
    foreach(GraphNode node, findChildren(current)){
      if(inClosedSet(node)!=NULL){
        //Nodo già presente
        continue;
      }
      else{
        if(inOpenSet(node)){
          //Nodo presente in openSet
          gValue=current->gValue()+current->distance(node);
          if(node->gValue()> gValue){
            node->setParent(current);
            node->setGValue(gValue);
```

```
            }
          }
          else{
            child = new AStarNode();
            child->setParent(current);
            child->setNode(node);
            distance=current->gValue()+current->distance(child);
            child->setGValue(distance);
            child->setHValue(child->distance(goalNode));
            //Aggiornamento openSet
            openSet->append(child);
          }
        }
      }
    }
}
```

4.4 Valutazione dei percorsi

Dopo aver trovato un insieme di path attraverso l'uso dell'algoritmo $AlphA^*$, come descritto nella sezione precedente, è necessario valutarli. Questa fase permette di estrarre il cammino più promettente, secondo la strategia scelta, assegnando un valore ad ogni percorso e scegliendo quello che risulta migliore. Questo valore viene calcolato attraverso una funzione di valutazione, diversa per ogni strategia implementata, che combina i criteri presi in considerazione.

Di seguito vengono descritte in dettaglio le tre strategie implementate.

4.4.1 MCDM-Pose

La prima strategia, chiamata in questa tesi *MCDM-Pose*, implementa il framework Multi-Criteria-Decision-Making proposto da Amigoni et al. [8] descritto in precedenza (Sezione 2.3). La funzione di valutazione combina un insieme di criteri con i rispettivi pesi, permettendo la valutazione di una possibile destinazione e mettendo in risalto le caratteristiche, ritenute più importanti, dei percorsi scelti per raggiungerla.

I criteri scelti per la valutazione dei cammini sono i seguenti:

- distance: rappresenta la lunghezza del percorso da valutare;

- battery: identifica il livello della batteria residua del robot, sottraendo una previsione del consumo dovuto alle rotazioni e traslazioni necessarie per seguire il percorso;
- information gain: rappresenta l'ampiezza della frontiera scelta come destinazione.

Questi criteri vengono comunicati al programma attraverso un file di configurazione letto all'avvio dell'applicazione, di cui segue un esempio.

```
<Criteria>
  <numOfCriteria>3</numOfCriteria>
  <singleCriterion>
    <name>distance</name>
        <weight>0.40</weight>
    <isActive>true</isActive>
  </singleCriterion>
  <singleCriterion>
    <name>informationGain</name>
        <weight>0.40</weight>
    <isActive>true</isActive>
  </singleCriterion>
  <singleCriterion>
    <name>battery</name>
        <weight>0.20</weight>
        <isActive>true</isActive>
  </singleCriterion>
    <combinedCriteria>
    <name>distance</name>
    <name>informationGain</name>
        <weight>0.95</weight>
  </combinedCriteria>
  <combinedCriteria>
    <name>distance</name>
    <name>battery</name>
    <weight>0.45</weight>
  </combinedCriteria>
  <combinedCriteria>
    <name>informationGain</name>
```

```
    <name>battery</name>
    <weight>0.7</weight>
  </combinedCriteria>
</Criteria>
```

Di seguito sono mostrate le funzioni che hanno il compito di calcolare i valori dei singoli criteri assegnati ai path presi in esame (Algoritmo 4.7) e di combinare i criteri così ottenuti, estraendo il cammino migliore (Algoritmo 4.8).

Algoritmo 4.7 Funzione di valutazione per una sola destinazione (MCDM-Pose).

```
double evaluateFrontier(Map map, int batteryTime){
  //Recupero percorsi verso la destinazione
  paths= prmAlgorithm->getPaths(Frontier);
  if(paths.empty()){
    //Nessun percorso valido
    foreach(Criterion c, activeCriteria){
      //Inserimento valore peggiore
      c->setWorstValue(frontier);
    }
  }
  else{
    //Calcolo valutazione percorsi
    foreach(Criterion c, activeCriteria){
      c->evaluate(frontier,paths,map,batteryTime);
    }
  }
}
```

Algoritmo 4.8 Funzione di valutazione di tutte le possibili destinazioni (MCDM-Pose).

```
EvaluationRecords evaluateFrontiers(Map map, int batteryTime){
  frontiers=map.frontiers();
  //Inizializzazione criteri
  foreach(Criterion criterion, criteria->values()){
    criterion->clean();
  }
  foreach(String name, matrix->getActiveCriteria()){
    activeCriteria->append(criteria->values(name));
  }
  //Valutazione destinazioni
  foreach(Frontier f, frontiers){
    evaluateFrontier(f,map,batteryTime);
  }
  //Normalizzazione criteri
  foreach(Criterion criterion, activeCriteria){
    criterion->normalize();
  }
  foreach(Frontier f, frontiers){
    foreach(Criterion criterion, activeCriteria){
      values=criterion->getEvaluation(f);
      frontierValues.append(values);
    }
    int bestPath =0;
    double bestValue=0.0;
    //Iterazione sui percorsi validi
    for(int j=0; j< pathNumber;j++){
      //Ordinamento criteri
      sort(activeCriteria,CriterionComparator(f,j));
      //Recupero valutazioni
      foreach(Criterion criterion, activeCriteria){
        values=criterion->getEvaluation(f);
        frontierValues.append(values);
      }
```

```
      lastCriterion= NULL;
      double final=0.0;
      //Calcolo valutazione
      for(int i=0; i<activeCriteria->length();i++){
        criterion=NULL;
        double weight=0.0;
        List<String> names;
        for(int k=i; k<activeCriteria->length();k++){
          next= activeCriteria->at(k);
          names.append(next->getName());
        }
        weight=matrix->getWeight(names);
        if(i==0){
          criterion=activeCriteria->first();
          temp= criterion->getEvaluation(f);
          final=final+temp.at(j)*weight;
        }
        else{
          criterion=activeCriteria->at(i);
          temp= criterion->getEvaluation(f);
          tempLast= lastCriterion->getEvaluation(f);
          double difference=temp.at(j)-tempLast.at(j);
          final=final+difference*weight;
        }
        lastCriterion=criterion;
      }
      if(final>bestValue){
        bestValue=final;
        bestPath=j;
      }
    }
    //Aggiornamento valutazioni
    eval->putEvaluation(f,bestValue);
  }
  double best=0.0;
  bestFrontier=NULL;
```

```
  //Calcolo destinazione migliore
  foreach(Frontier f, eval->getFrontiers()){
    double v = eval->getEvaluation(f);
    if(v>best){
      best=v;
      bestFrontier= f;
    }
  }
  if(bestFrontier!=NULL){
    planner->calculateActions(bestFrontier,map);
  }
  return eval;
}
```

4.4.2 Tovar et al.

Questa strategia, chiamata in questa tesi *Tovar*, implementa parzialmente la soluzione proposta da Tovar et al. [18] descritta in precedenza (Sezione 2.2). Per adattare questa strategia al codice esistente, è stato necessario modificare la funzione di valutazione originale, che utilizzava informazioni impossibili da ottenere con la nostra implementazione. La funzione di valutazione proposta (Equazione 4.4.1), misura la qualità di un possibile percorso, stimandone il guadagno d'informazione e la precisione necessaria per seguirlo, tralasciando però i parametri che identificano i landmark e gli spigoli, presenti nella versione originale (Equazione 2.2.1).

$$T_i = \sum_{i=1}^{m} \left(e^{(l_{v_i} - s_{v_i})} \prod_{j=1}^{q_i} \left(\frac{e^{-|\theta_j|}}{\sqrt{s_j} + 1} \right) \right) \tag{4.4.1}$$

- i posizione in cui viene eseguita una scansione lungo il path;
- m numero totale di operazioni di rilevamento lungo il path;
- l_{v_i}lunghezza della frontiera più vicina alla posizione i;
- s_{v_i}distanza della prossima posizione i al centro della frontiera più vicina;
- j indice delle posizioni assunte dal robot proseguendo lungo il path;
- q_i numero totale di stop del robot per raggiungere la posizione i;

- θ_j cambiamento di orientamento per raggiungere la configurazione successiva j;
- s_j distanza tra la posizione j e la posizione successiva $j+1$.

I criteri utilizzati da questa strategia possono essere semplificati come segue:

- ampiezza delle rotazioni effettuate;
- lunghezza del path da valutare;
- ampiezza della frontiera più vicina ad ogni posizione intermedia del percorso.

Di seguito sono mostrate le funzioni che hanno il compito di calcolare i valori assegnati ai path verso la destinazione presa in esame (Algoritmo 4.9) e di estrarre il cammino migliore (Algoritmo 4.10).

Algoritmo 4.9 Funzione di valutazione per una sola destinazione (Tovar).

```
double evaluateFrontier(Frontier frontier, Map map,int batteryTime){
  //Recupero percorsi verso la destinazione
  paths= prmAlgorithm->getPaths(frontier));
  if(paths.empty()){
    //Nessun percorso valido
    frontierValue->insert(frontier,0);
  }
  else{
    //Esistono percorsi validi
    robotPose=map.lastRobotPose();
    pathsList->insert(frontier,paths);
    //Calcolo valutazione percorsi
    foreach(Path path, paths){
      double pathValue= 0;
      int m=path.length();
      //Iterazione sui passi del percorso
      for(int i=0;i<m;i++){
        point=path.at(i);
        bestFrontier= NULL;
```

```
      double distance= INFINITY;
      //Ricerca frontiera più vicina
      foreach(Frontier frontier, map.frontiers()){
        double d=point->distance(frontier);
        if(d<distance){
          distance=d;
          bestFrontier=frontier;
        }
      }
      double lvi=bestFrontier->length();
      double svi =0;
      if(i<m-1){
        svi=path.at(i+1)->distance(bestFrontier);
      }
      double temp=exp(lvi-svi);
      double prod=1;
      double theta= robotPose->getTheta();
      for(int j=0; j<i;j++){
        point1= path.at(j);
        point2= path.at(j+1);
        poseIni(point1),theta);
        poseFin(point2,0);
        double oj=computeRotation(poseIni,poseFin);
        double sj=point1->distance(point2);
        double temp2= exp(-oj)/(sqrt(sj)+1);
        theta=theta+oj;
        prod=prod*temp2;
      }
      temp=temp*prod;
      pathValue=pathValue+temp;
    }
    values.append(pathValue);
  }
  //Aggiornamento valori
  frontierValue->insert(frontier,values);
}
```

```
  return values;
}
```

Algoritmo 4.10 Funzione di valutazione di tutte le possibili destinazioni (Tovar).

```
EvaluationRecords evaluateFrontiers(Map map, int batteryTime){
  frontiers=map.frontiers();
  //Valutazione frontiere
  foreach(Frontier f, frontiers){
    evaluateFrontier(f,map,batteryTime);
  }
  //Ricerca percorso migliore per ogni frontiera
  foreach(Frontier f, frontiers){
    //Recupero valori percorsi
    values= frontierValue->value(f);
    int bestPath =0;
    double bestValue=0;
    for(int i=0; i<values.size();i++){
      double value=values.at(i);
      if(value>bestValue){
        //Trovato un percorso migliore
        bestValue=value;
        bestPath=i;
      }
    }
    //Salvataggio valutazione e percorso
    eval->putEvaluation(f,bestValue);
    bestPathList->insert(f,path);
  }
  double best=0;
  bestFrontier=NULL;
  //Ricerca destinazione migliore
  foreach(Frontier f, eval->getFrontiers()){
    double value = eval->getEvaluation(f);
    if(value>best){
```

```
      //Trovata destinazione migliore
      best=value;
      bestFrontier= f;
    }
    if(bestFrontier!=NULL){
      //Esiste una destinazione
      //Calcolo delle azioni per raggiungerla
      planner->calculateActions(bestFrontier,map);
    }
    return eval;
  }
}
```

4.4.3 MCDM-Path

Questa strategia, chiamata in questa tesi *MCDM-Path*, utilizza il framework Multi-Criteria-Decision-Making [8] come nella versione MCDM-Pose descritta in precedenza (Sezione 4.4.1), ma con criteri diversi. Tale variante implementa una funzione di valutazione simile alla precedente, che combina un insieme di criteri con i rispettivi pesi, permettendo la valutazione di una possibile destinazione e mettendo in risalto le caratteristiche, ritenute più importanti, dei percorsi scelti per raggiungerla. Questa strategia si propone come una fusione delle politiche descritte in precedenza, valutando l'informazion gain ottenuto lungo tutto il path utilizzando il framework MCDM.

I criteri scelti per la valutazione dei cammini sono i seguenti:

- distance: rappresenta la lunghezza del percorso da valutare;
- battery: identifica il livello della batteria residua del robot, sottraendo una previsione del consumo dovuto alle rotazioni e traslazioni necessarie per seguire il percorso;
- information gain: rappresenta l'ampiezza delle frontiere visibili lungo l'intero cammino scelto. A differenza della strategia MCDM-Pose, che valuta esclusivamente l'informazione ottenuta raggiungendo la destinazione, con la strategia MCDM-Path l'information gain è ottenuta sommando le porzioni di frontiere visibili dalla posizione del robot con i sensori in suo possesso (Figura 4.6).

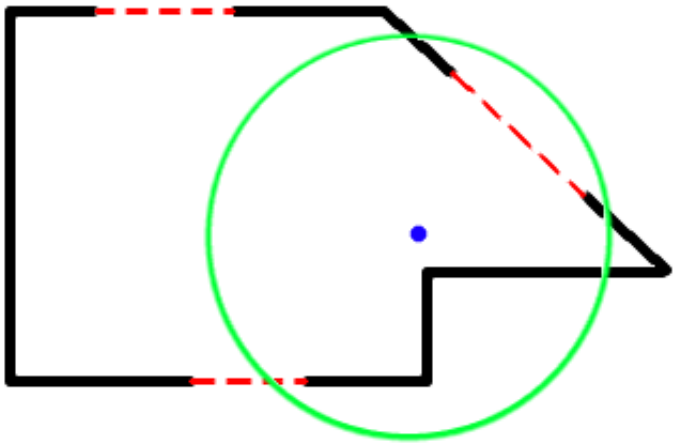

Figura 4.6: Calcolo delle porzioni di frontiere visibili dalla posizione del robot (MCDM-Path). Il cerchio verde rappresenta la portata del sensore laser.

Questi criteri vengono combinati attraverso l'uso della funzione di utilità MCDM (Equazione 2.3.1) (Algoritmo 4.11), calcolando i valori dei singoli criteri iterando su tutti i punti che compongono il path (Algoritmo 4.12).

Come nella versione MCDM-Pose, questi criteri vengono comunicati al programma attraverso un file di configurazione letto all'avvio dell'applicazione.

Algoritmo 4.11 Funzione di valutazione di tutte le possibili destinazioni (MCDM-Path).

```
EvaluationRecords evaluateFrontiers(Map map, int batteryTime){
  frontiers=map.frontiers();
  //Inizializzazione criteri
  foreach(Criterion criterion, criteria->values()){
    criterion->clean();
  }
  foreach(String name, matrix->getActiveCriteria()){
    activeCriteria->append(criteria->value(name));
  }
  //Valutazione destinazioni
  foreach(Frontier f, frontiers){
    evaluateFrontier(f,map,batteryTime,powerSignalData);
  }
```

```
//Normalizzazione criteri
foreach(Criterion criterion, activeCriteria){
  criterion->normalize();
}
foreach(Frontier f, frontiers){
  foreach(Criterion criterion, activeCriteria){
    value=criterion->getEvaluation(f);
    frontierValues.append(value);
  }
  int bestPath =0;
  double bestValue=0.0;
  //Iterazione sui percorsi validi
  for(int j=0; j< pathNumber;j++){
    //Ordinamento criteri
    sort(activeCriteria,CriterionComparator(f,j));
    //Recupero valutazioni
    foreach(Criterion criterion, activeCriteria){
      value=criterion->getEvaluation(f);
      frontierValues.append(value);
    }
    lastCriterion= NULL;
    double final=0.0;
    //Calcolo valutazione
    for(int i=0; i<activeCriteria->length();i++){
      Criterion criterion=NULL;
      double weight=0.0;
      List<String> names;
      for(int k=i; k<activeCriteria->length();k++){
        next= activeCriteria->at(k);
        names.append(next->getName());
      }
      weight=matrix->getWeight(names);
      if(i==0){
        criterion=activeCriteria->first();
        temp= criterion->getEvaluation(f);
        final=final+temp.at(j)*weight;
```

```
        }
        else{
          criterion=activeCriteria->at(i);
          temp= criterion->getEvaluation(f);
          tempLast= lastCriterion->getEvaluation(f);
          double difference=temp.at(j)-tempLast.at(j);
          final=final+difference*weight;
        }
        lastCriterion=criterion;
      }
      if(final>bestValue){
        bestValue=final;
        bestPath=j;
      }
    }
    //Aggiornamento valutazioni
    eval->putEvaluation(f,bestValue);
  }
  double best=0.0;
  bestFrontier=NULL;
  //Calcolo destinazione migliore
  foreach(Frontier f, eval->getFrontiers()){
    double v = eval->getEvaluation(f);
    if(v>best){
      best=v;
      bestFrontier= f;
    }
  }
  if(bestFrontier!=NULL){
    planner->calculateActions(bestFrontier,map);
  }
  return eval;
}
```

Algoritmo 4.12 Funzione di valutazione per una sola destinazione (MCDM-Path).

```
double evaluateFrontier(Frontier frontier, Map map, int batteryTime){
  //Recupero percorsi verso la destinazione
  paths= prmAlgorithm->getPaths(frontier);
  if(paths.empty()){
    //Nessun percorso valido
    foreach(Criterion c, activeCriteria){
      c->setWorstValue(frontier);
    }
  }
  else{
    //Calcolo valutazione percorsi
    foreach(Criterion c, activeCriteria){
      c->evaluate(frontier,paths,map,batteryTime);
    }
  }
}
```

Capitolo 5

Risultati sperimentali

In questo capitolo vengono illustrate le prove sperimentali svolte per valutare il sistema sviluppato in questo lavoro di tesi e avere un riscontro della principale idea di partenza, ovvero sull'effetto dell'utilizzo dei path come criterio per l'esplorazione. Queste prove, inoltre, hanno dato la possibilità di verificare se la nostra applicazione, messa a confronto con le strategie MCDM-Pose e Tovar et al., introduca effettivamente miglioramenti.

Nonostante il simulatore USARSim utilizzato metta a disposizione diverse tipologie di robot, la nostra scelta è ricaduta sui modelli *Pioneer 3 AT* (P3AT) che rappresentano i robot più stabili implementati e sono stati utilizzati nel codice esistente alla base del nostro lavoro. Per determinare le caratteristiche di questi agenti robotici, è necessario modificare un file di configurazione nel simulatore, chiamato *UDKUSAR.ini*, che contiene tutti i modelli di agenti esistenti nel simulatore ed ogni sensore presente sul robot.

I modelli P3AT sono stati equipaggiati con un telemetro laser che simula il funzionamento di un sensore *SICK* reale. La gittata del laser è stata impostata a 20 metri, la sua risoluzione angolare a 1° e il Field of View (FOV) a 360°. Nonostante l'impossibilità tecnica di utilizzare un singolo laser con questo FOV, il suo comportamento può essere ottenuto, nella pratica, con l'uso di due telemetri laser posizionati sul robot alla stessa altezza, uno in verso di avanzamento e uno in verso opposto, ciascuno con un FOV a 180°. Oltre ai sensori laser, i P3AT sono equipaggiati con due cinture di sonar, cinque anteriori e cinque posteriori che sono utilizzati per rilevare gli ostacoli in prossimità del robot, presenti all'interno del mondo simulato, troppo bassi per essere percepiti dal sensore laser. E' presente, inoltre, il sensore di Ground Truth che può fornire al controllore la vera posizione del robot nella mappa (e che usualmente non

ha alcuna controparte reale). Questo sensore è stato introdotto dagli sviluppatori di USARSim per eliminare gli errori di misura che potrebbero influenzare la stima della posizione del robot, permettendo così la valutazione di metodi di localizzazione alternativi. Infine il robot è equipaggiato con una telecamera, che riceve immagini dal simulatore, utilizzata per ottenere le immagini dell'ambiente virtuale.

Come detto in precedenza (Sezione 3.1), il simulatore USARSim si basa sulla piattaforma UDK, utilizzata nella versione di maggio 2012, la stessa presa in considerazione dal team PoAReT, per i propri test. In aggiunta, sempre sul repository SourceForge di USARSim[1], è stato scaricato il Wireless Simulation Server (WSS). Per questo componente, nelle nostre simulazioni, è sempre stata utilizzata la modalità NoopPropagationModel[2]. Dal momento che la piattaforma UDK esiste esclusivamente in versione per Windows, il computer utilizzato per le simulazioni e gli esperimenti è stato equipaggiato con il sistema operativo Windows 7.

Le caratteristiche rilevanti del computer usato per i test sono:

- CPU: Intel® Core™ i5 3550 3,7 GHz;
- RAM: 8 GB DDR3 1600 MHz;
- Scheda Grafica: NVIDIA® GeForce® GTX 660, 2 Gb di RAM dedicata.

Ogni esperimento in cui sono stati fissati i parametri per poterlo eseguire (l'ambiente, la posizione di partenza del robot, la strategia adottata, i criteri scelti, ...), viene definito *run*. Per lo svolgimento delle prove sperimentali abbiamo fissato un limite di tempo di 20 minuti per ogni run. Al raggiungimento di tale limite gli esperimenti sono stati fermati e le performance dei robot sono state memorizzate in opportuni file MATLAB. Inoltre sono stati scritti dei particolari script, utilizzando il programma MATLAB, che si occupano dell'estrazione di dati, immagini e valori che interessano la valutazione del nostro lavoro di tesi (per esempio: l'area mappata, la distanza percorsa dal robot, ...). Gli esperimenti sono stati eseguiti variando, per ogni mappa utilizzata, la strategia adottata ed il punto considerato come posizione di partenza del robot, durante la simulazione. In riferimento alle politiche di esplorazione, le

[1] sourceforge.net/projects/usarsim

[2] In questa modalità il componente WSS simula un modello di comunicazione wireless senza tener conto di eventuali ostacoli.

prove sono state eseguite variando tra le seguenti funzioni di valutazione: la nostra strategia MCDM-Path (Capitolo 4), MCDM-Pose (Sezione 2.3) (utilizzata come strategia di riferimento dal team PoAReT) e Tovar et al. (Sezione 2.2) (ad oggi l'unica strategia basata sulla valutazione dell'intero path).

Le strategie basate su MCDM sono caratterizzate da un'elevata flessibilità, dovuta alla possibilità di specificare i pesi da assegnare ai singoli criteri e ad ogni possibile loro combinazione. Nella tabella proposta (Tabella 5.1), sono stati riportati i criteri ed i pesi utilizzati, in questo lavoro di tesi, per l'esecuzione dei test.

Criteri	**Pesi**
Distance	0.40
Information gain	0.40
Battery	0.20
Distance, information gain	0.95
Distance, battery	0.45
Battery, information gain	0.70

Tabella 5.1: Tabella rappresentante la combinazione di criteri e pesi adottati per il framework MCDM.

5.1 Mappe utilizzate

I test sono stati condotti su diverse mappe, ognuna con caratteristiche ben precise. Al fine di testare il sistema in diverse situazioni e condizioni, sono state preparate ed utilizzate, inizialmente, due mappe dalle caratteristiche simili (VMAC2011 (Figura 5.1) e la VMAC2012 (Figura 5.2)).

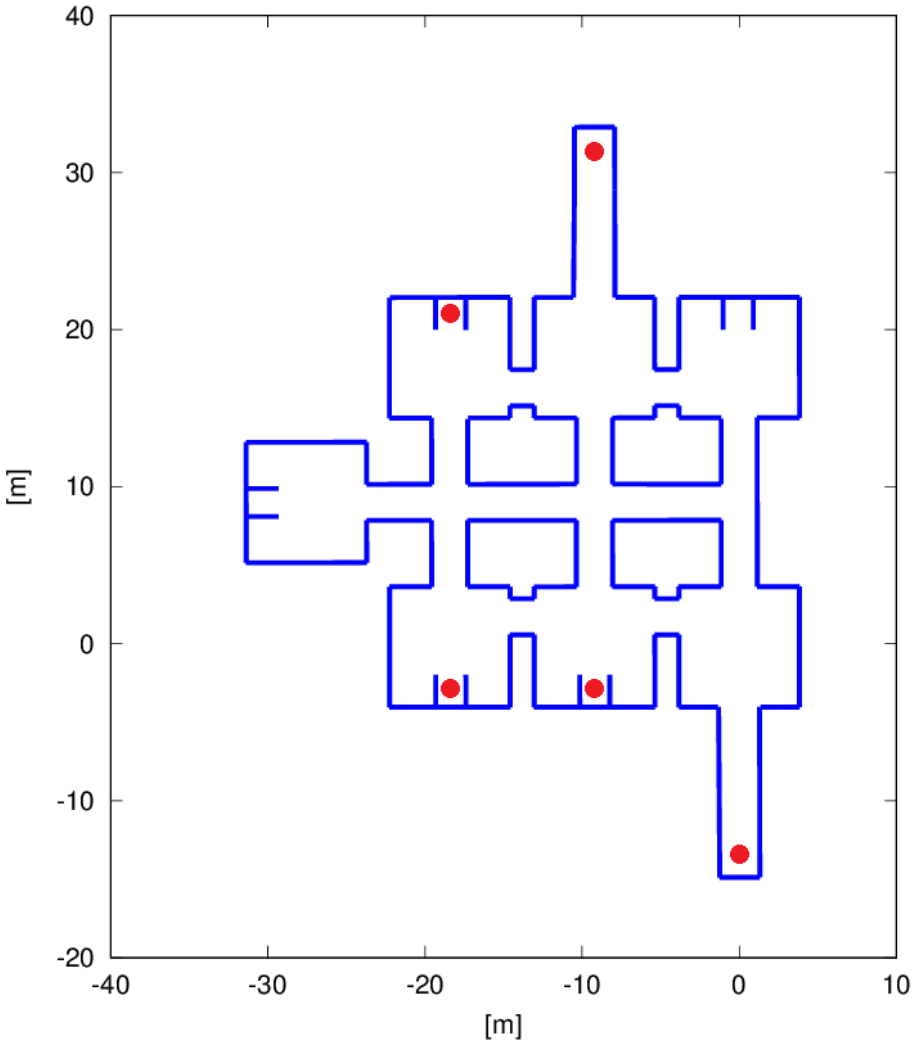

Figura 5.1: Rappresentazione dell'ambiente VMAC2011 di area 609.75 m^2. I punti rossi rappresentano le posizioni iniziali che può assumere il robot durante i test sperimentali, la stessa convenzione è stata utilizzata nelle figure successive.

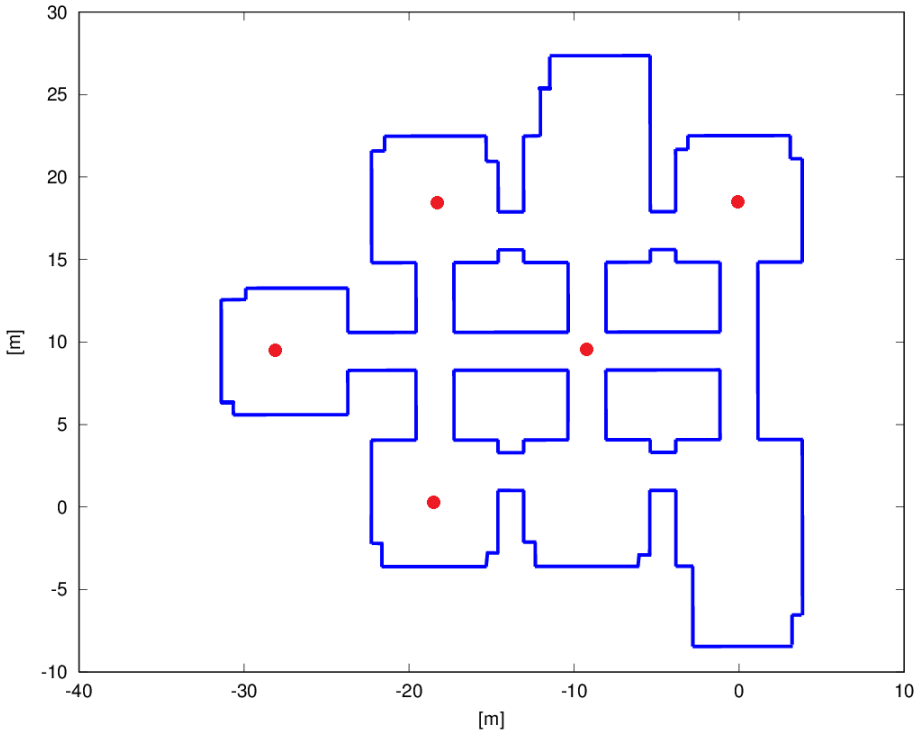

Figura 5.2: Rappresentazione dell'ambiente VMAC2012 di area 601.25 m^2.

Successivamente si è deciso di adottare altri ambienti e di considerarne solo una porzione, estraendoli dalla mappa Orio[3] (Figura 5.3) e dalla mappa VascheLibraryFloor1[4] presente nei repository Radish[5] (Figura 5.4), ottenute da ambienti realmente esistenti e già adottate in precedenza dal team PoAReT.

[3]http://www.oriocenter.it
[4]http://cres.usc.edu/radishrepository/outgoing/vasche_library_floor1/oldlibrary.gif
[5]http://radish.sourceforge.net

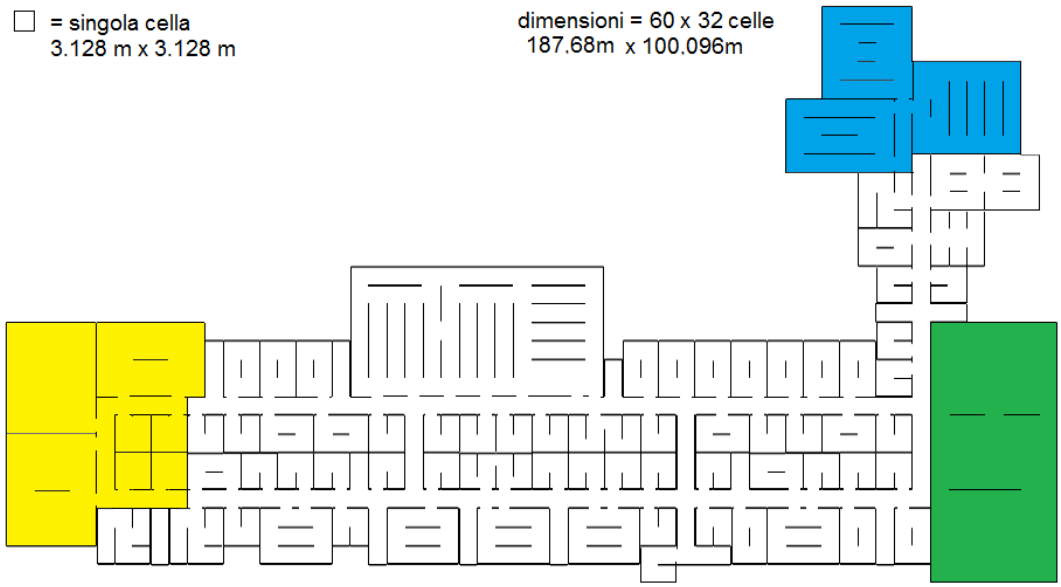

Figura 5.3: Rappresentazione dell'ambiente del primo piano del centro commerciale Orio Center utilizzata dal team PoAReT. Questa mappa è stata poi divisa per permettere i test con un singolo robot. La zona verde identifica la parte di mappa denominata *OpenSpace*, la zona gialla rappresenta la parte di mappa chiamata *Mix* e la zona azzurra identifica la parte di mappa in seguito definita *Obstacle*.

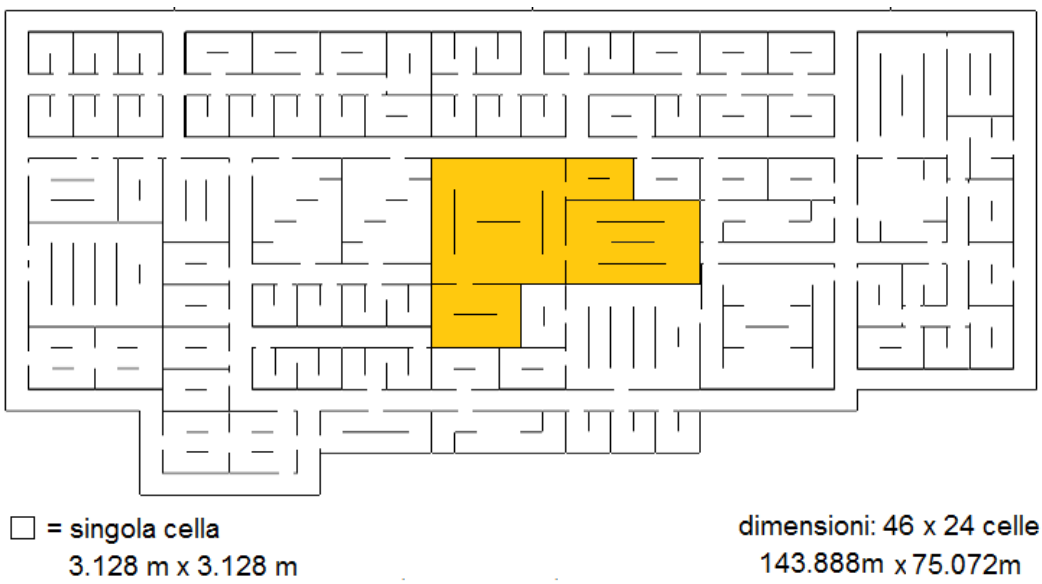

Figura 5.4: Rappresentazione dell'ambiente VascheLibraryFloor1 utilizzato dal team PoAReT. Questa mappa è stata poi divisa per permettere i test su un singolo robot. La zona arancione identifica la parte di mappa utilizzata nei test Radish.

E' stata presa la decisione di dividere questi ambienti, in quanto le mappe in oggetto erano state studiate e create per permettere un'esplorazione multi-robot, risultando troppo vaste per il nostro sistema basato su agente singolo. Per rendere i test più significativi, queste sotto-mappe sono state scelte appositamente per rappresentare ambienti con caratteristiche diverse tra loro, cercando di ottenere diversi tipologie di ambientazioni:

- una mappa caratterizzata da ampi spazi aperti (OpenSpace (Figura 5.21));
- due ambienti con un elevato numero di ostacoli e spazi chiusi (Obstacle (Figura 5.30) e Radish (Figura 5.48));
- una variante che rappresenta una via di mezzo delle precedenti (Mix (Figura 5.39)).

5.2 Misura delle prestazioni

L'obiettivo della ricerca di un oggetto in ambienti inizialmente sconosciuti e senza informazioni sulla sua posizione, può essere equiparato al massimizzare l'area esplorata nel minor tempo possibile. La scarsità delle informazioni iniziali sull'ambiente non permette un'immediata ottimizzazione della strategia di esplorazione; per ovviare a questo problema, una possibile opzione è quella di sfruttare le informazioni ottenute incrementalmente del robot per un'ottimizzazione greedy. Si può osservare che non tutti i criteri usati nelle strategie di esplorazione considerate (il consumo di batteria, l'information gain, la distanza percorsa, ...) sono direttamente collegati ad una misura di performance globale. Quello che ci si aspetta di ottenere, cercando di ottimizzare tutti i criteri, è un migliore risultato finale rispetto a quello che si otterrebbe ottimizzandone esclusivamente uno solo.

Le misure di performance (Tabella 5.2) che sono state utilizzate per la valutazione del sistema sono:

- l'area misurata ad un orizzonte temporale, definito dal tempo (medio) di completamento della mappatura da parte della strategia MCDM-Path;
- il tempo impiegato per mappare una percentuale fissata della mappa, tipicamente l'80-90%. La scelta di questa percentuale ha senso in applicazioni USAR, dove l'importante è mappare l'area dell'ambiente velocemente, tralasciando se necessario gli angoli e le zone meno significative.

Misura della performance	Lavori
Area esplorata nel tempo	[8], [3]
Tempo impiegato per mappare una percentuale di area	[2], [22], [21]

Tabella 5.2: Misure delle prestazioni e lavori dello stato dell'arte in cui sono state utilizzate.

Si noti che alcune delle misure di prestazioni elencate in precedenza, sono molto soggette a fluttuazioni poiché fortemente dipendenti dalla mappa e dal singolo esperimento. Al fine di validare gli esperimenti da un punto di vista statistico, ogni mappa scelta è stata testata fino a dieci volte per ogni strategia di esplorazione implementata. In questo modo è stato possibile ottenere un campione significativo di esecuzioni per poter calcolare, per ogni misura, un valore medio e una deviazione standard non elevata. Per validare il confronto tra i risultati ottenuti, sono stati effettuati dei test *ANOVA* con una soglia di significatività pari a 0.05.

5.3 Risultati ottenuti

In questa sezione vengono presentati i risultati ottenuti con l'esecuzione dei test sperimentali per le strategie descritte nel capitolo precedente (Sezione 4.4). Per la valutazione verranno utilizzate le misure delle performance descritte in precedenza (Sezione 5.2). Per ogni mappa sono mostrati dei grafici comparativi delle tre strategie utilizzate, con dei commenti riguardanti gli esiti ottenuti e la loro rilevanza sulla conferma o meno dell'ipotesi iniziale di questo lavoro di tesi. Inoltre, come ulteriore verifica, sono stati messi a confronto diversi path per osservare il comportamento qualitativo assunto dalle tre strategie utilizzate (Sezione 5.3.5).

5.3.1 Mappa VMAC2011

Questa mappa (Figura 5.1), utilizzata nella Virtual Manufacturing Automation Competition (VMAC) del 2011, presenta stanze di medie dimensioni con una scarsa presenza di ostacoli, unite da corridoi facilmente percorribili dal robot.

Le figure riportate presentano i risultati delle dieci prove eseguite per ognuna delle tre strategie implementate: MCDM-Path (Figura 5.5), MCDM-Pose (Figura 5.6) e Tovar (Figura 5.7). La linea rossa punteggiata mostra la media dei dieci test effettuati.

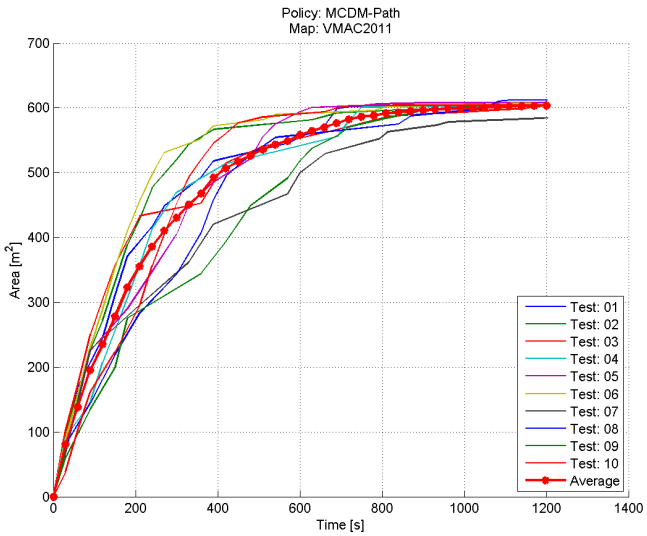

Figura 5.5: Grafico che rappresenta i risultati dei test eseguiti sulla mappa VMAC2011 con la strategia MCDM-Path.

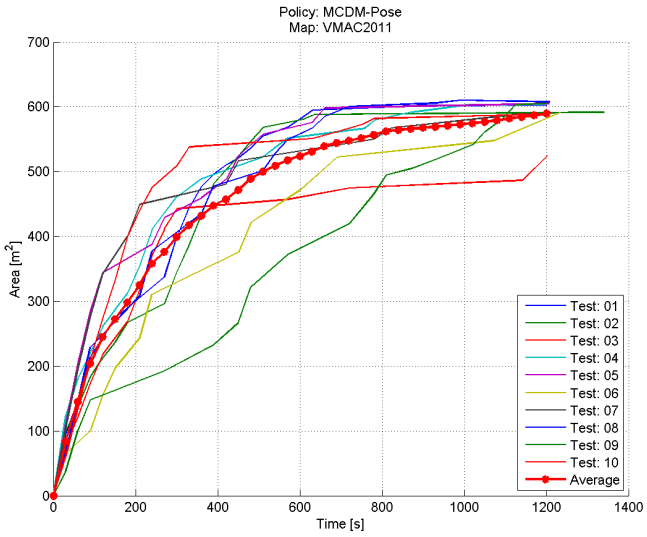

Figura 5.6: Grafico che rappresenta i risultati dei test eseguiti sulla mappa VMAC2011 con la strategia MCDM-Pose.

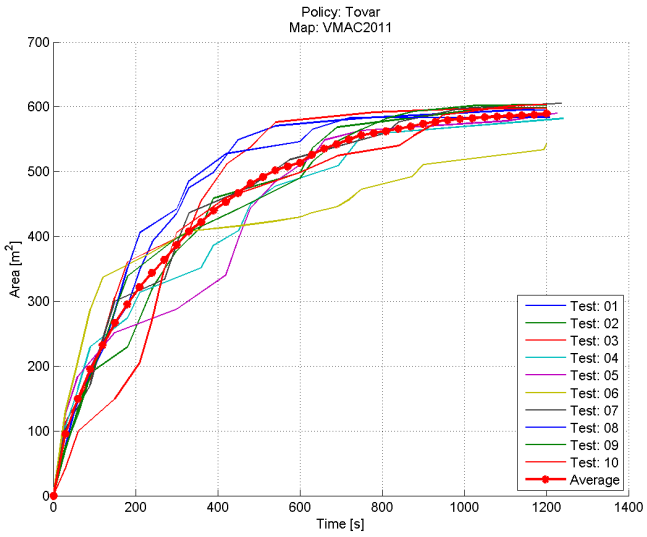

Figura 5.7: Grafico che rappresenta i risultati dei test eseguiti sulla mappa VMAC2011 con la strategia di Tovar.

Da come si evince dai grafici sopra riportati, possiamo affermare che l'andamento delle prove sperimentali di MCDM-Path sembrerebbe più regolare rispetto alle altre strategie da noi implementate.

Nei grafici seguenti sono state confrontate le medie e le deviazioni standard (Figura 5.8) ottenute elaborando i risultati dei grafici precedenti.

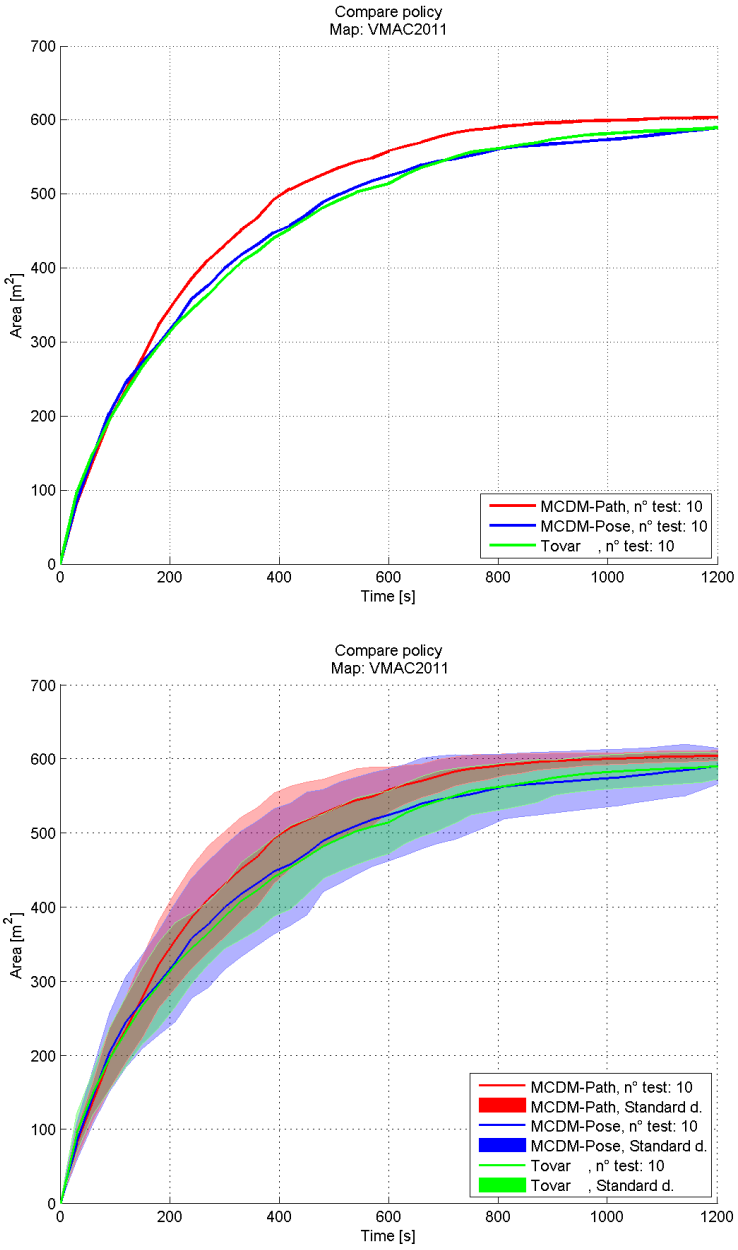

Figura 5.8: In alto è mostrato il confronto tra le medie dei risultati ottenuti con i dieci test delle tre strategie implementate sulla mappa VMAC2011, mentre in basso si può notare il confronto tra le deviazioni standard.

Per ottenere dei risultati statisticamente rilevanti, la sola analisi dei grafici precedenti non è sufficiente. Per un confronto più significativo tra le strategie adottate è stato eseguito un test *ANOVA* con soglia di significatività α pari a 0.05. Per mostrare in modo evidente che le differenze nei grafici precedenti non sono solo frutto del caso, è necessario che i p-value risultanti dalle tabelle seguenti siano inferiori al valore α.

Confronto dell'area misurata ad un orizzonte temporale fissato (960s).

- confronto con la strategia MCDM-Pose
  - i risultati completi sono riportati nella tabella 5.3 mentre nell'immagine 5.9 sono presentati in forma grafica

Source	SS	df	MS	F	p-value
Columns	4805	1	4805	4,09	**0,0583**
Error	21153	18	1175	-	-
Total	25958	19	-	-	-

Tabella 5.3: Risultati del test ANOVA sull'area misurata della mappa VMAC2011, dalle strategie MCDM-Path e MCDM-Pose, ad un orizzonte temporale di 960s.
Il significato delle voci in tabella è il seguente: **SS** è la somma dei quadrati, **df** sono i gradi di libertà, **MS** è la varianza, **F** è la statistica del test. La riga **Columns** indica che i valori precedenti sono stati calcolati valutando la varianza tra i gruppi, mentre nella riga **Error** valutando la varianza interna ai gruppi. La riga **Total** contiene le somme dei valori in colonna. La stessa convenzione è utilizzata nelle tabelle che seguono.

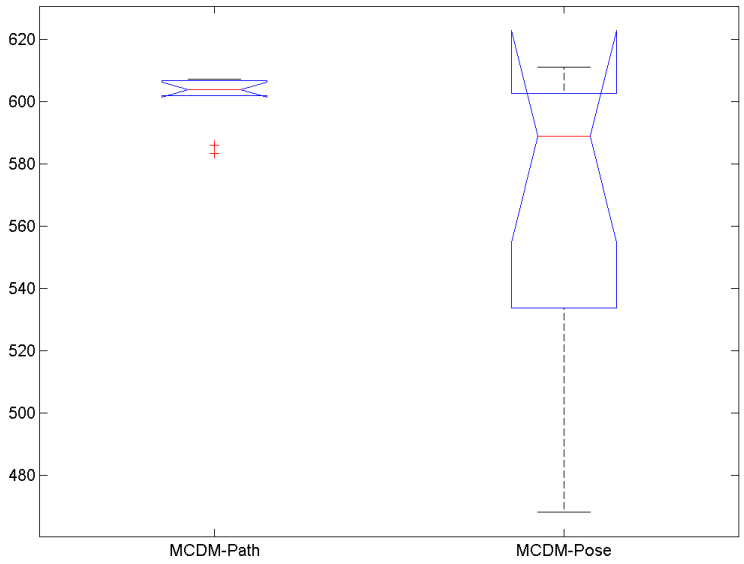

Figura 5.9: Risultati del test ANOVA sull'area misurata della mappa VMAC2011, dalle strategie MCDM-Path e MCDM-Pose, ad un orizzonte temporale di 960s.

- confronto con la strategia Tovar
  - la tabella 5.4 mostra i risultati del test *ANOVA* sul confronto tra le strategie MCDM-Path e Tovar. Nell'immagine 5.10 i risultati sono presentati in forma grafica

Source	**SS**	**df**	**MS**	**F**	**p-value**
Columns	1526	1	1526	4.20	**0.0553**
Error	6545	18	363	-	-
Total	8071	19	-	-	-

Tabella 5.4: Risultati del test ANOVA sull'area misurata della mappa VMAC2011, dalle strategie MCDM-Path e Tovar, ad un orizzonte temporale di 960s.

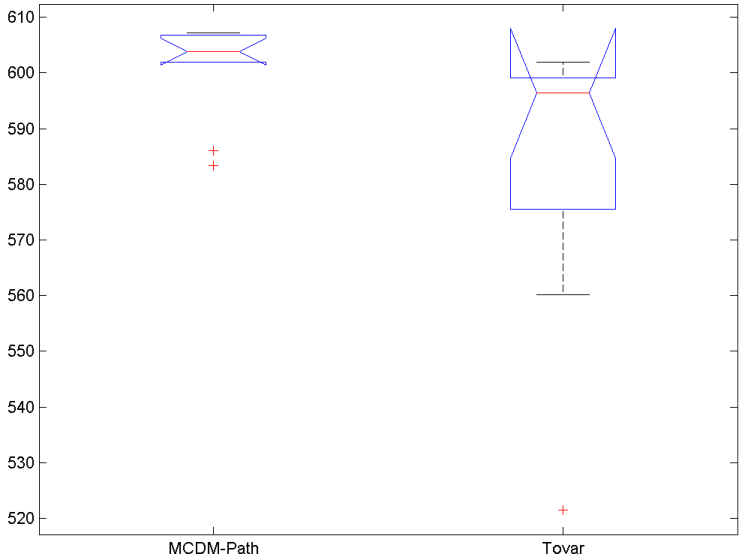

Figura 5.10: Risultati del test ANOVA sull'area misurata della mappa VMAC2011, dalle strategie MCDM-Path e Tovar, ad un orizzonte temporale di 960s.

Dai test eseguiti sulla varianza, risulta che MCDM-Path non ha prestazioni significativamente migliori rispetto a Tovar o a MCDM-Pose.

Confronto del tempo impiegato per mappare l'80% dell'area.

- confronto con la strategia MCDM-Pose
  - i risultati del test *ANOVA* sono riportati nella tabella 5.5 e nell'immagine 5.11

Source	SS	df	MS	F	p-value
Columns	55125	1	55125	1.24	**0.2796**
Error	798570	18	44365	-	-
Total	853695	19	-	-	-

Tabella 5.5: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa VMAC2011, da parte dalle strategie MCDM-Path e MCDM-Pose.

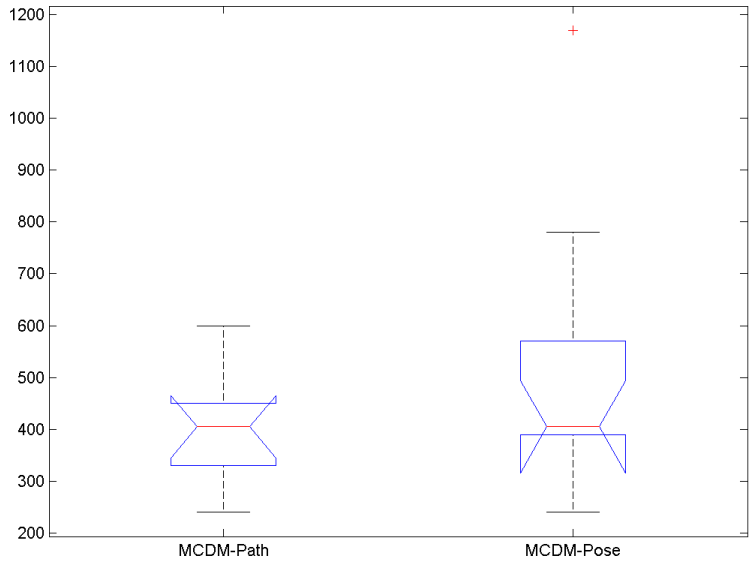

Figura 5.11: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa VMAC2011, da parte dalle strategie MCDM-Path e MCDM-Pose.

- confronto con la strategia Tovar
  - i risultati ottenuti sono riportati nella tabella 5.6 in forma testuale, mentre nell'immagine 5.12 sono presentati in forma grafica

Source	**SS**	**df**	**MS**	**F**	**p-value**
Columns	61605	1	61605	2.80	**0.1117**
Error	396450	18	22025	-	-
Total	458055	19	-	-	-

Tabella 5.6: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa VMAC2011, da parte dalle strategie MCDM-Path e Tovar.

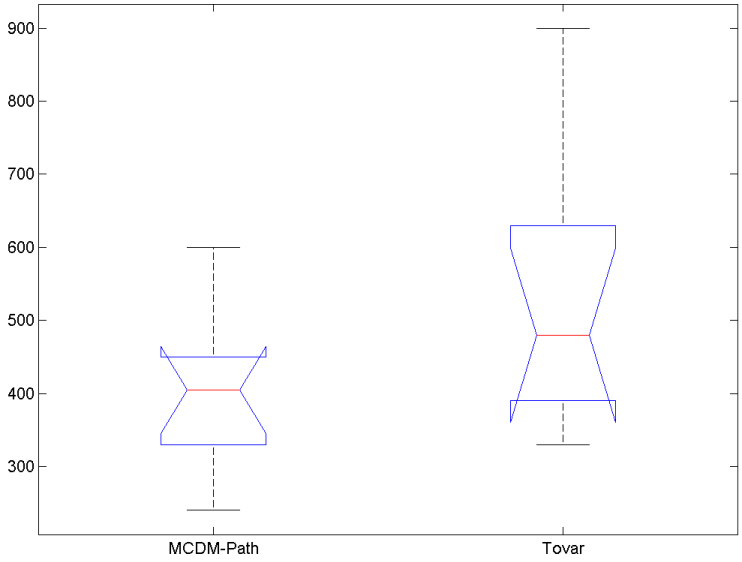

Figura 5.12: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa VMAC2011, da parte dalle strategie MCDM-Path e Tovar.

Dai risultati sopra riportati possiamo dire che, nonostante guardando i grafici possa sembrare che la strategia MCDM-Path risulti migliore rispetto alle alternative, un'analisi più approfondita della varianza rivela che la differenza tra le tre strategie non è statisticamente rilevante in questo tipo di ambiente.

5.3.2 Mappa VMAC2012

Questa mappa (Figura 5.2), utilizzata nella Virtual Manufacturing Automation Competition (VMAC) del 2012, presenta stanze di medie dimensioni senza la presenza di ostacoli, unite da corridoi facilmente percorribili dal robot. Questo ambiente è molto simile al precedente (VMAC2011) e rappresenta la stessa tipologia di mappa.

Di seguito sono mostrati i risultati delle dieci prove divisi in base alla strategia utilizzata: MCDM-Path (Figura 5.13), MCDM-Pose (Figura 5.14) e Tovar (Figura 5.15). Come nel caso precedente, la linea rossa punteggiata mostra la media dei test effettuati.

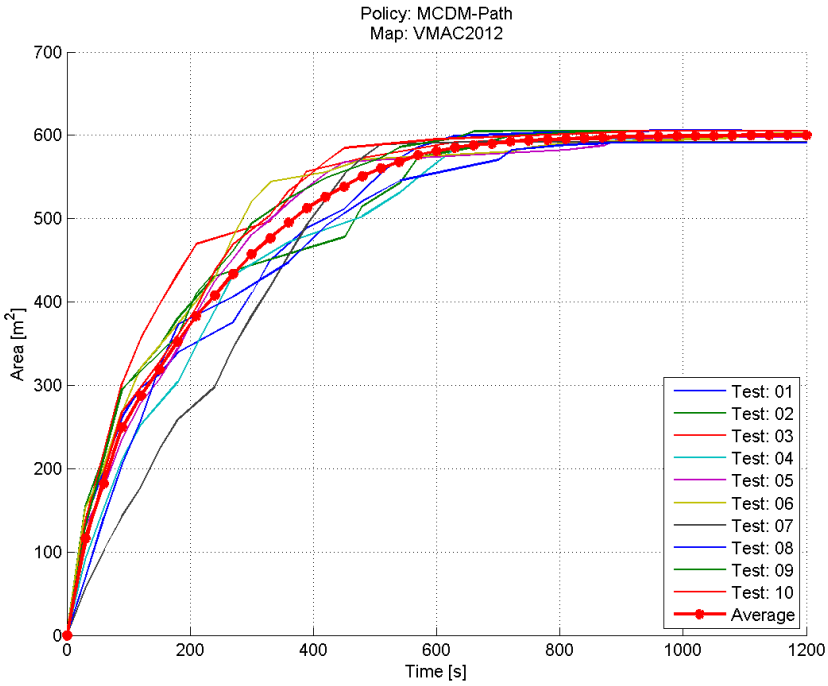

Figura 5.13: Grafico che rappresenta i risultati dei test eseguiti sulla mappa VMAC2012 con la strategia MCDM-Path.

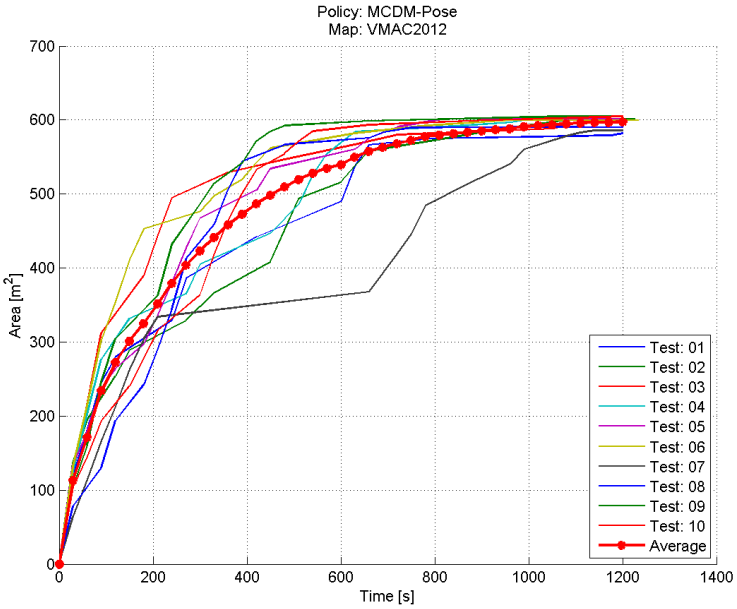

Figura 5.14: Grafico che rappresenta i risultati dei test eseguiti sulla mappa VMAC2012 con la strategia MCDM-Pose.

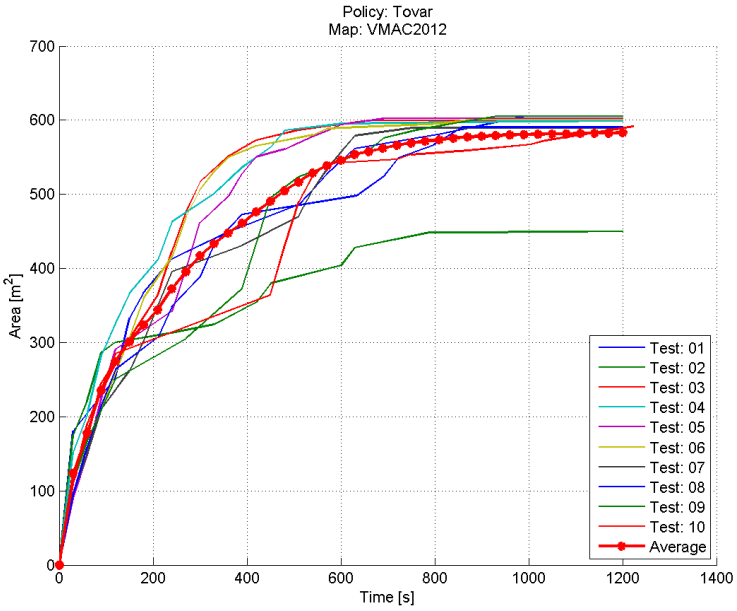

Figura 5.15: Grafico che rappresenta i risultati dei test eseguiti sulla mappa VMAC2012 con la strategia di Tovar.

Elaborando i risultati riportati nei grafici precedenti, è stato effettuato il confronto tra le medie e le deviazioni standard (Figura 5.16).

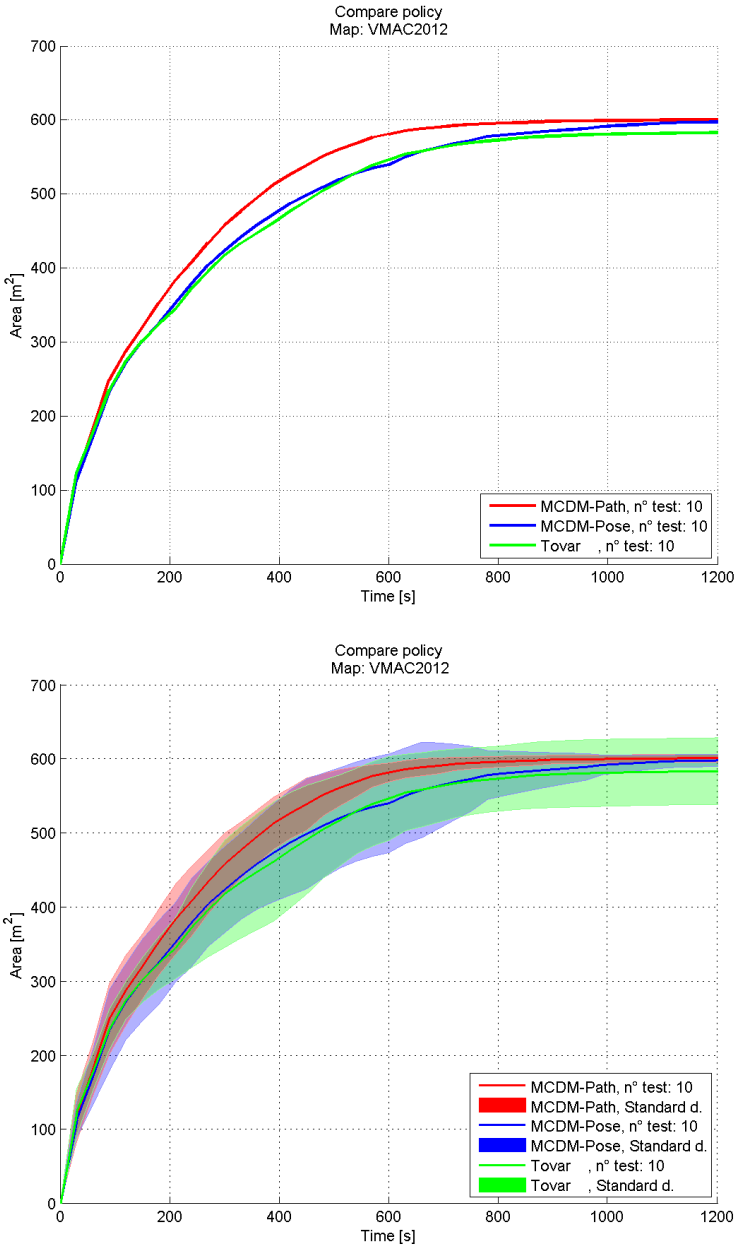

Figura 5.16: In alto è mostrato il confronto tra le medie dei risultati ottenuti con i dieci test delle tre strategie implementate nella mappa VMAC2012, mentre in basso si può notare il confronto tra le deviazioni standard.

Come nel caso precedente è stato eseguito un test *ANOVA*, con soglia di significatività pari a 0.05, per mostrare in modo evidente se le differenze nei grafici precedenti sono causate da una reale differenza di prestazioni delle tre strategie implementate.

Confronto dell'area misurata ad un orizzonte temporale di 960s.

- confronto con la strategia MCDM-Pose
  - i risultati completi sono riportati, nella tabella 5.7 in forma testuale, mentre in forma grafica nell'immagine 5.17

Source	SS	df	MS	F	p-value
Columns	236	1	236	2.12	**0.1626**
Error	2004	18	111	-	-
Total	2240	19	-	-	-

Tabella 5.7: Risultati del test ANOVA sull'area misurata della mappa VMAC2012, dalle strategie MCDM-Path e MCDM-Pose, ad un orizzonte temporale di 960s.

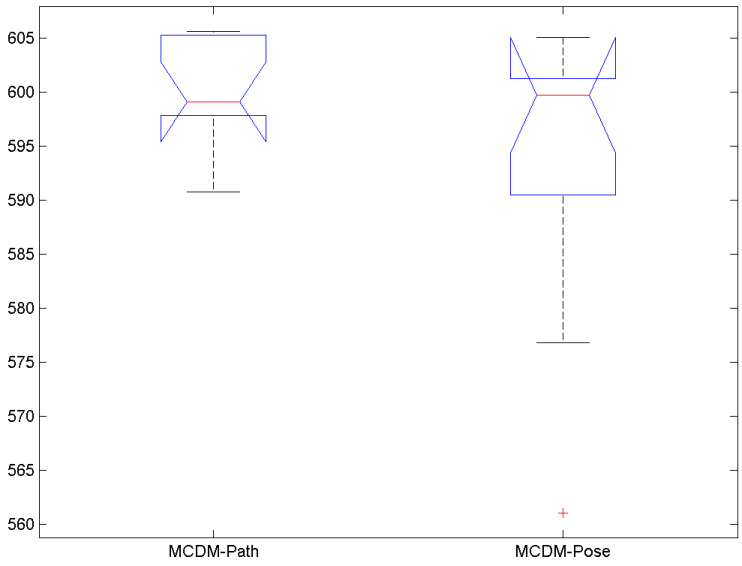

Figura 5.17: Risultati del test ANOVA sull'area misurata della mappa VMAC2012, dalle strategie MCDM-Path e MCDM-Pose, ad un orizzonte temporale di 960s.

- confronto con la strategia Tovar
  - il test *ANOVA* presenta i risultati mostrati dalla tabella 5.8 e dall'immagine 5.18

Source	SS	df	MS	F	p-value
Columns	1953	1	1953	1.70	**0.2089**
Error	20694	18	1149	-	-
Total	22647	19	-	-	-

Tabella 5.8: Risultati del test ANOVA sull'area misurata della mappa VMAC2012, dalle strategie MCDM-Path e Tovar, ad un orizzonte temporale di 960s.

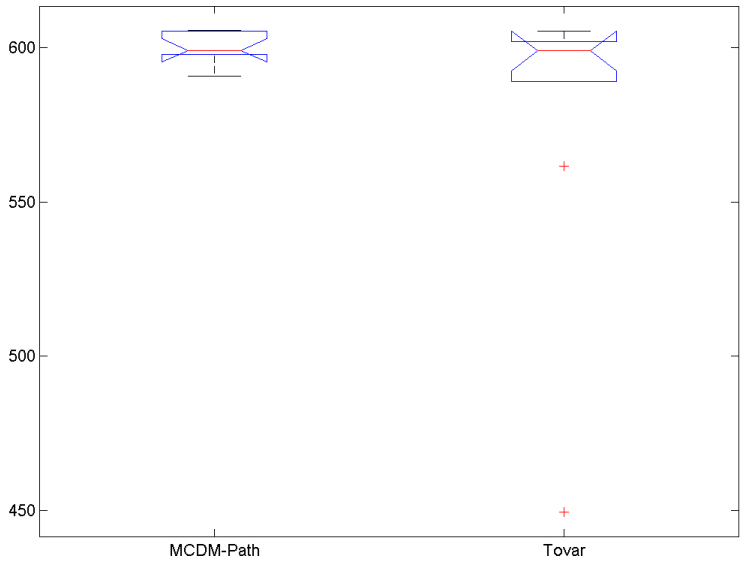

Figura 5.18: Risultati del test ANOVA sull'area misurata della mappa VMAC2012, dalle strategie MCDM-Path e Tovar, ad un orizzonte temporale di 960s.

Confronto del tempo impiegato per mappare l'80% dell'area.

- confronto con la strategia MCDM-Pose

  - la tabella 5.9 mostra i risultati del test *ANOVA*, mentre una rappresentazione grafica è mostrata nell'immagine 5.19

Source	SS	df	MS	F	p-value
Columns	23805	1	23805	1.52	**0.2341**
Error	282690	18	15705	-	-
Total	306495	19	-	-	-

Tabella 5.9: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa VMAC2012, da parte dalle strategie MCDM-Path e MCDM-Pose.

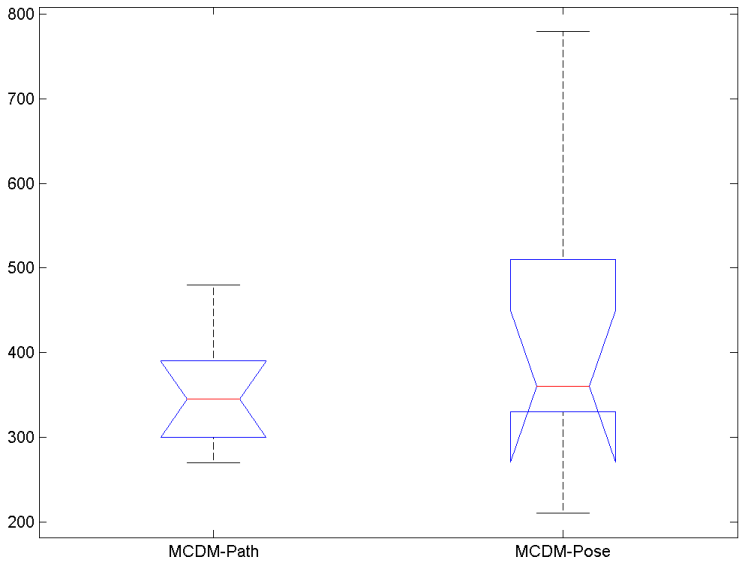

Figura 5.19: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa VMAC2012, da parte dalle strategie MCDM-Path e MCDM-Pose.

- confronto con la strategia Tovar
  - come nel confronto precedente, i risultati sono riportati sia in forma testuale nella tabella 5.10 che in forma grafica (Figura 5.20)

Source	SS	df	MS	F	p-value
Columns	103680	1	103680	2.68	**0.1187**
Error	695340	18	38630	-	-
Total	799020	19	-	-	-

Tabella 5.10: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa VMAC2012, da parte dalle strategie MCDM-Path e Tovar.

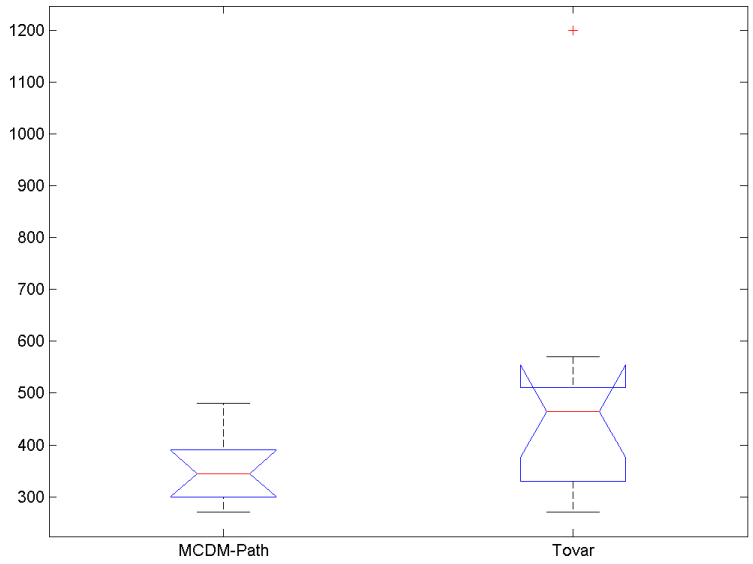

Figura 5.20: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa VMAC2012, da parte dalle strategie MCDM-Path e Tovar.

Come nella mappa precedente, i test eseguiti su questo ambiente non permettono di determinare con sicurezza quale sia la strategia migliore. Questo risultato non sorprende visto le similitudini tra le due mappe VMAC. Sebbene infatti, la strategia MCDM-Path risulti migliore osservando i grafici delle medie, ancora una volta un'analisi più approfondita della varianza rivela che la differenza tra le tre strategie non è statisticamente rilevante. E' possibile però notare che il comportamento del robot, utilizzando la strategia MCDM-Path (Figura 5.13), risulta più costante durante l'esplorazione se confrontato con i metodi alternativi (Figura 5.14) (Figura 5.15), grazie alla capacità di ottimizzare gli spostamenti, massimizzando le informazioni raccolte durante l'intero cammino seguito.

5.3.3 Mappa Orio completa

Da questa mappa (Figura 5.3), proveniente dal lavoro originale su cui si basa questa tesi, sono stati estratti tre ambienti con caratteristiche differenti, adatti all'esplorazione con singolo agente.

5.3.3.1 OpenSpace

La mappa OpenSpace (Figura 5.21), estratta dall'ambiente Orio, è caratterizzata da ampi spazi aperti che favoriscono le percezioni del robot. La scarsità di ostacoli rende molto semplice il movimento dell'agente all'interno dell'area.

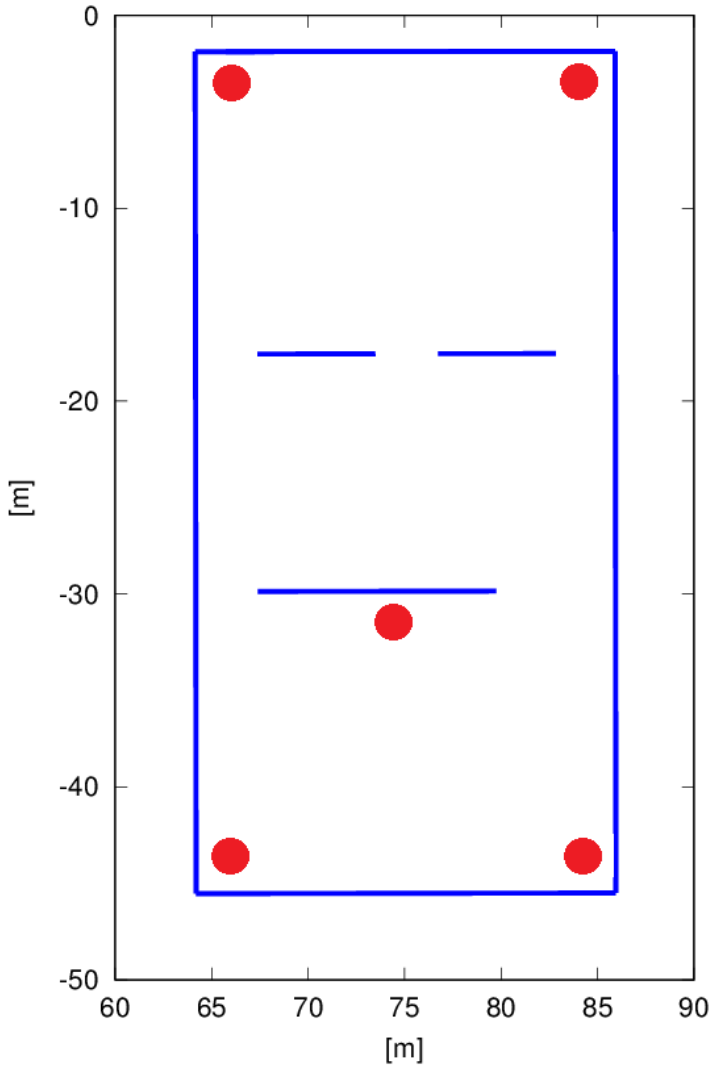

Figura 5.21: Rappresentazione dell'ambiente Orio OpenSpace di area 945.75 m^2.

Le figure seguenti presentano i risultati delle dieci prove eseguite con le tre strategie implementate: MCDM-Path (Figura 5.22), MCDM-Pose (Figura 5.23) e Tovar (Figura 5.24) con evidenziata la media dei test effettuati.

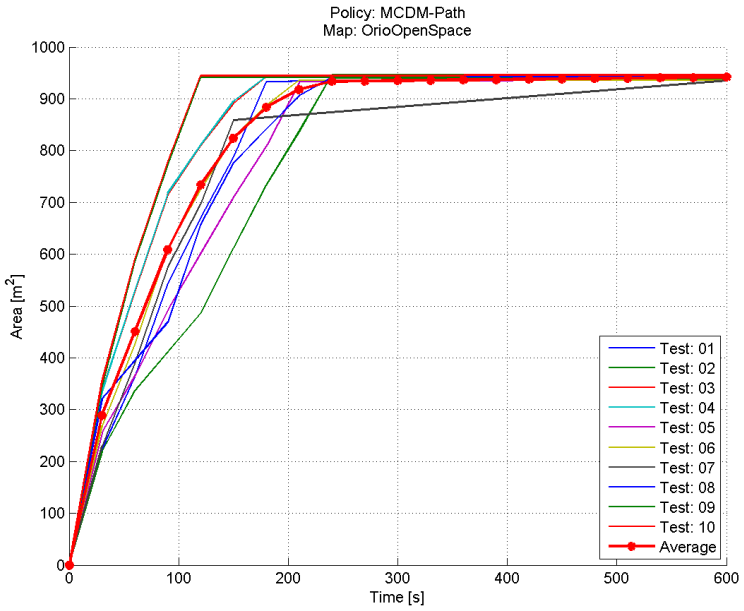

Figura 5.22: Grafico che rappresenta i risultati dei test eseguiti sulla mappa OpenSpace con la strategia MCDM-Path.

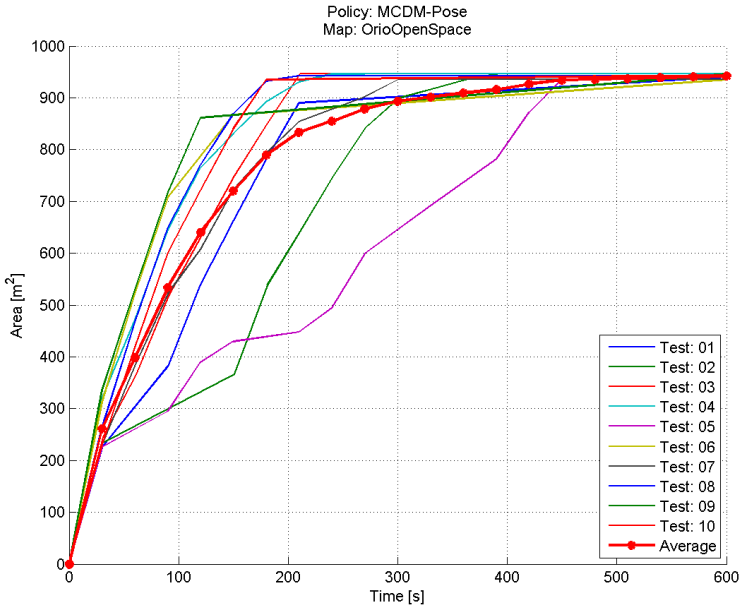

Figura 5.23: Grafico che rappresenta i risultati dei test eseguiti sulla mappa OpenSpace con la strategia MCDM-Pose.

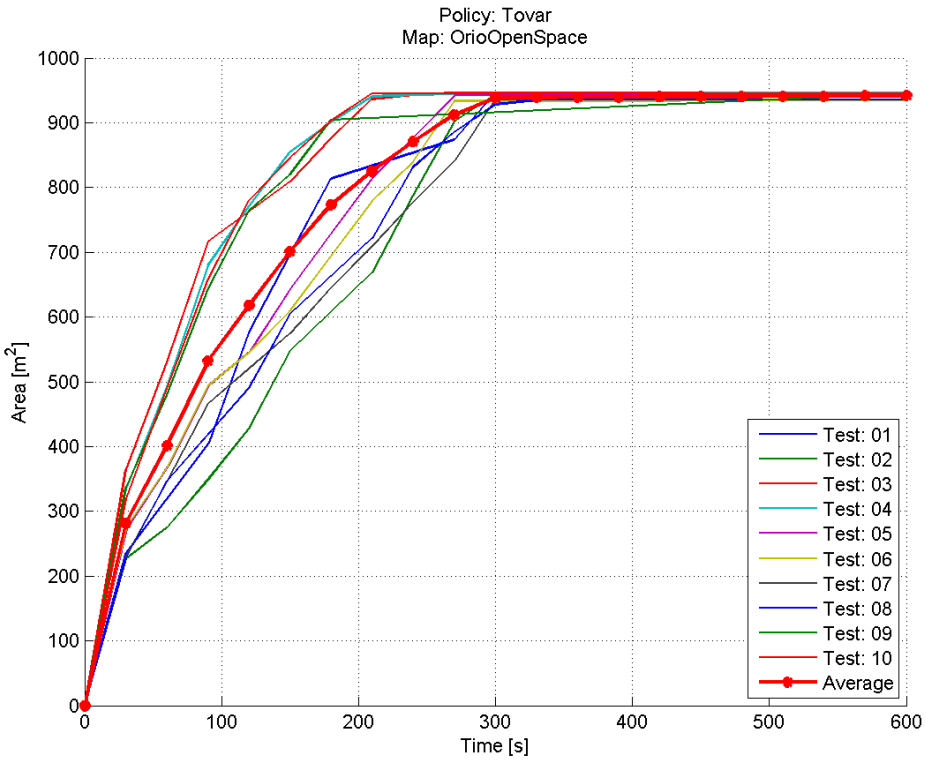

Figura 5.24: Grafico che rappresenta i risultati dei test eseguiti sulla mappa OpenSpace con la strategia di Tovar.

Nei grafici seguenti sono state confrontate le medie e le deviazioni standard (Figura 5.25) ottenute elaborando i risultati dei test eseguiti sulla mappa OpenSpace.

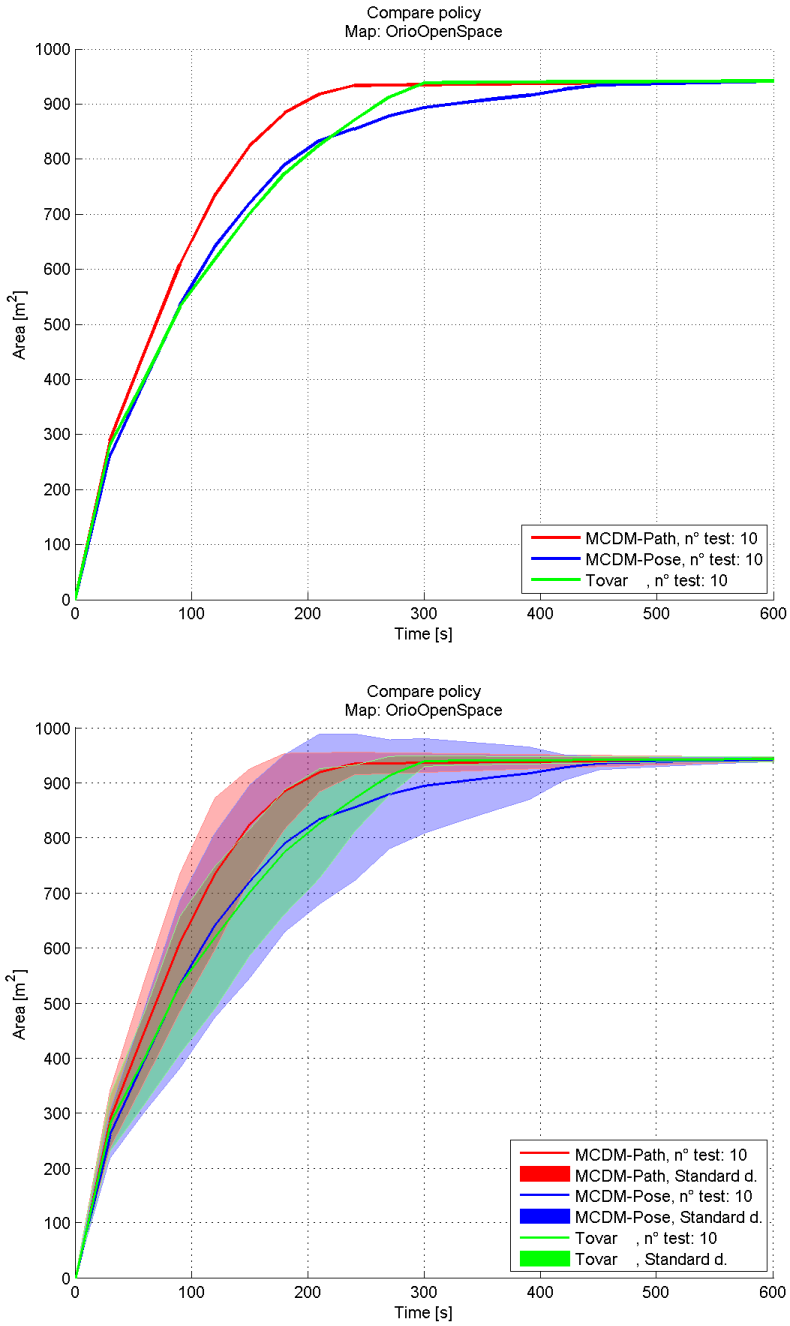

Figura 5.25: In alto è mostrato il confronto tra le medie dei risultati ottenuti con i dieci test sulla mappa OpenSpace delle tre strategie implementate, mentre in basso si può notare il confronto tra le deviazioni standard.

Anche in questo caso è stato eseguito un test *ANOVA* con soglia di significatività α pari a 0.05, per ottenere dei risultati statisticamente rilevanti.

Confronto dell'area misurata ad un orizzonte temporale di 240s.

- confronto con la strategia MCDM-Pose
  - i risultati completi sono riportati nella tabella 5.11 mentre nell'immagine 5.26 sono presentati in forma grafica

Source	SS	df	MS	F	p-value
Columns	14936	1	14936	1.43	**0.2474**
Error	188071	18	10448	-	-
Total	203008	19	-	-	-

Tabella 5.11: Risultati del test ANOVA sull'area misurata della mappa OpenSpace, dalle strategie MCDM-Path e MCDM-Pose, ad un orizzonte temporale di 240s.

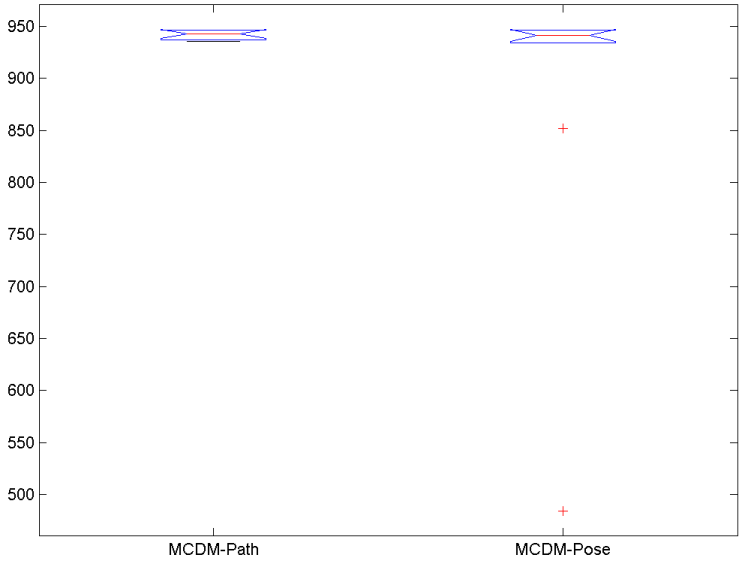

Figura 5.26: Risultati del test ANOVA sull'area misurata della mappa OpenSpace, dalle strategie MCDM-Path e MCDM-Pose, ad un orizzonte temporale di 240s.

- confronto con la strategia Tovar
  - i risultati ottenuti con il test *ANOVA*, sono riportati nella tabella 5.12 e nella figura 5.27

Source	SS	df	MS	F	p-value
Columns	2332	1	2332	1.85	**0.1903**
Error	22664	18	1259	-	-
Total	24997	19	-	-	-

Tabella 5.12: Risultati del test ANOVA sull'area misurata della mappa Open-Space, dalle strategie MCDM-Path e Tovar, ad un orizzonte temporale di 240s.

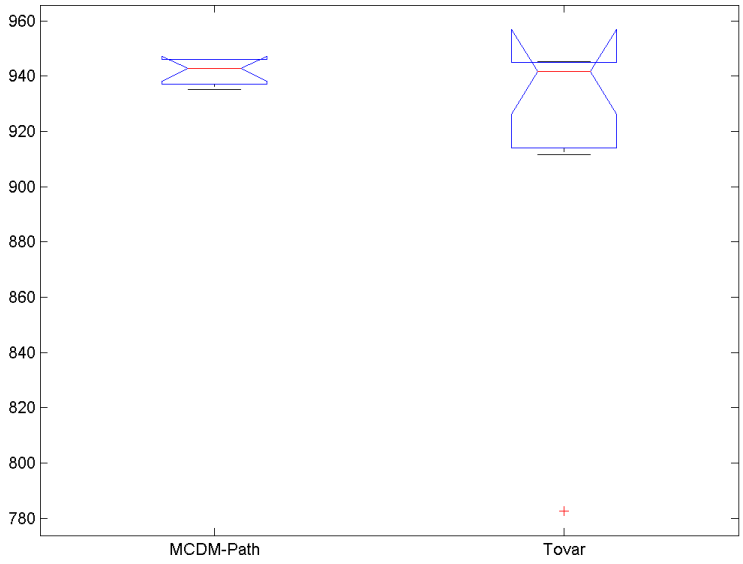

Figura 5.27: Risultati del test ANOVA sull'area misurata della mappa Open-Space, dalle strategie MCDM-Path e Tovar, ad un orizzonte temporale di 240s.

Confronto del tempo impiegato per mappare il 90% dell'area.

- confronto con la strategia MCDM-Pose
  - come nei test precedenti, i risultati sono riportati in forma testuale (Tabella 5.13) e in forma grafica (Figura 5.28)

Source	SS	df	MS	F	p-value
Columns	11520	1	11520	2.88	**0.1069**
Error	72000	18	4000	-	-
Total	83520	19	-	-	-

Tabella 5.13: Risultati del test ANOVA sul tempo impiegato per mappare il 90% dell'area della mappa OpenSpace, da parte dalle strategie MCDM-Path e MCDM-Pose.

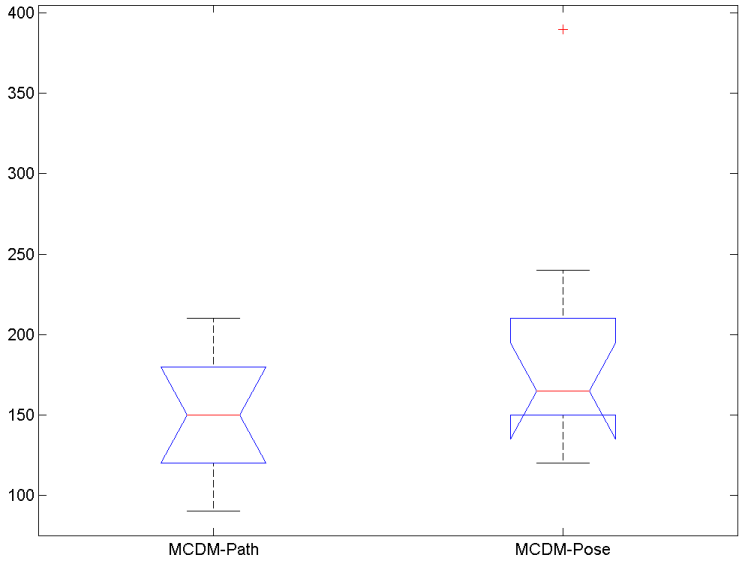

Figura 5.28: Risultati del test ANOVA sul tempo impiegato per mappare il 90% dell'area della mappa OpenSpace, da parte dalle strategie MCDM-Path e MCDM-Pose.

- confronto con la strategia Tovar
  - i risultati completi sono riportati nella tabella 5.14 mentre nell'immagine 5.29 si mostrano i risultati in forma grafica

Source	SS	df	MS	F	p-value
Columns	19845	1	19845	12.21	**0.0026**
Error	29250	18	1625	-	-
Total	49095	19	-	-	-

Tabella 5.14: Risultati del test ANOVA sul tempo impiegato per mappare il 90% dell'area della mappa OpenSpace, da parte dalle strategie MCDM-Path e Tovar.

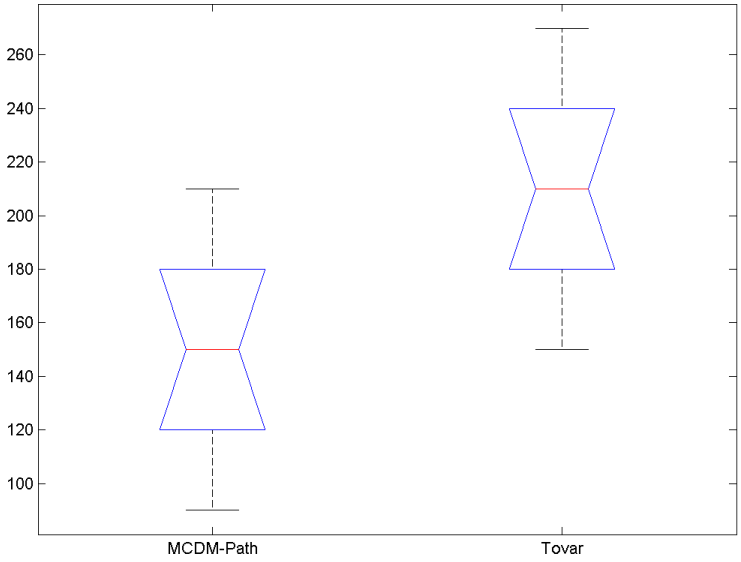

Figura 5.29: Risultati del test ANOVA sul tempo impiegato per mappare il 90% dell'area della mappa OpenSpace, da parte dalle strategie MCDM-Path e Tovar.

Analizzando i risultati ottenuti, si può notare che in questa mappa, caratterizzata da ampi spazi aperti e da una scarsa presenza di ostacoli, nessuna delle strategie implementate prevale in modo evidente sulle altre. Questo è dovuto alla capacità del robot di percepire ampi spazi della mappa da qualunque posizione dell'ambiente, riducendo così i vantaggi che uno studio più approfondito dei path offre, come nelle strategie MCDM-Path e Tovar.

5.3.3.2 Obstacle

La mappa Obstacle (Figura 5.30), estratta dall'ambiente Orio, è caratterizzata da tre grandi stanze con la presenza di numerosi ostacoli che impediscono

la visione completa della stanza senza muoversi al suo interno.

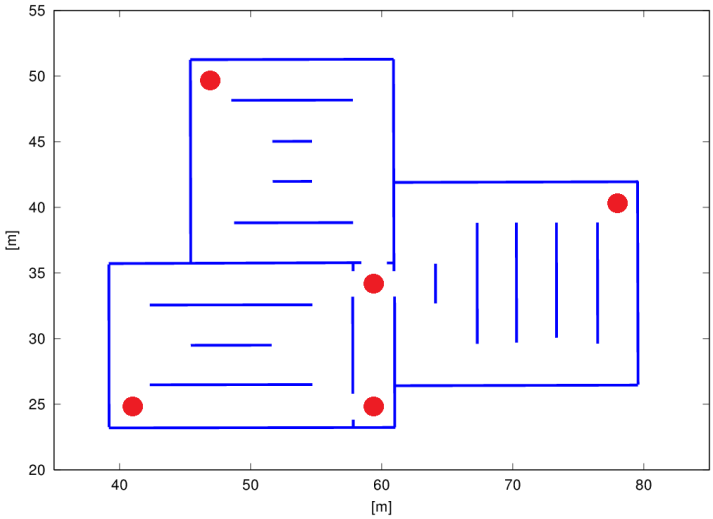

Figura 5.30: Rappresentazione dell'ambiente Orio Obstacle di area 803.5 m^2.

I risultati delle dieci prove eseguite con le tre strategie implementate, sono mostrati nelle immagini seguenti (MCDM-Path (Figura 5.31), MCDM-Pose (Figura 5.32) e Tovar (Figura 5.33)). La linea rossa punteggiata mostra la media dei dieci test effettuati.

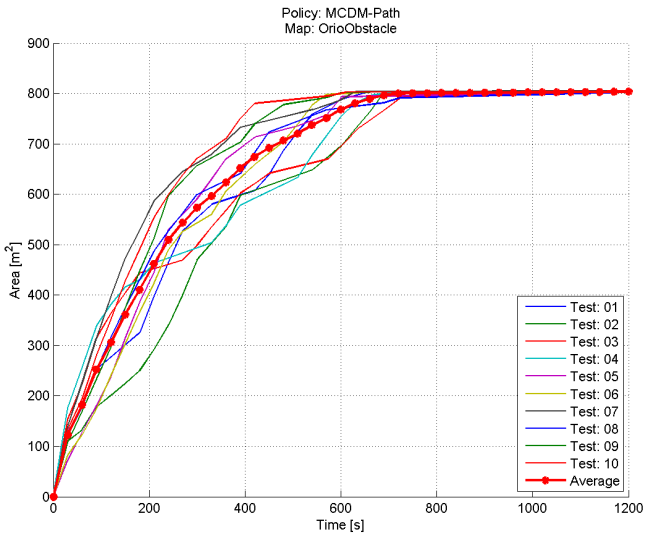

Figura 5.31: Grafico che rappresenta i risultati dei test eseguiti sulla mappa Obstacle con la strategia MCDM-Path.

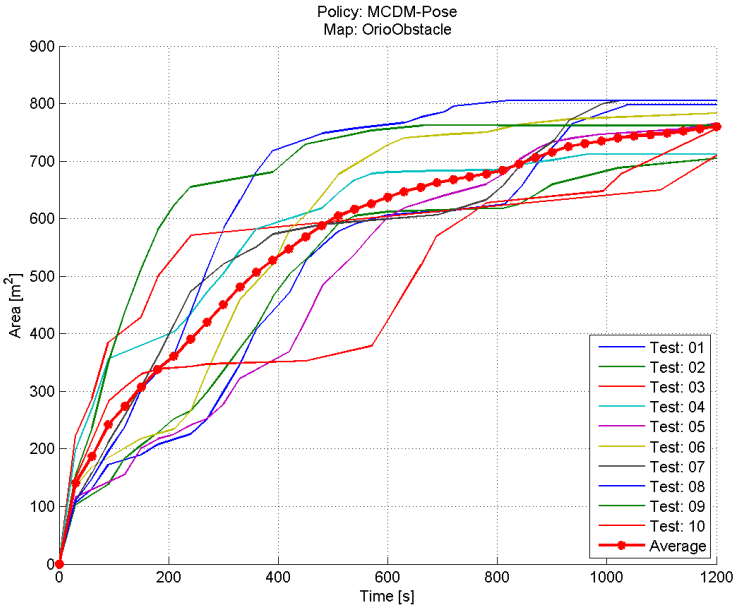

Figura 5.32: Grafico che rappresenta i risultati dei test eseguiti sulla mappa Obstacle con la strategia MCDM-Pose.

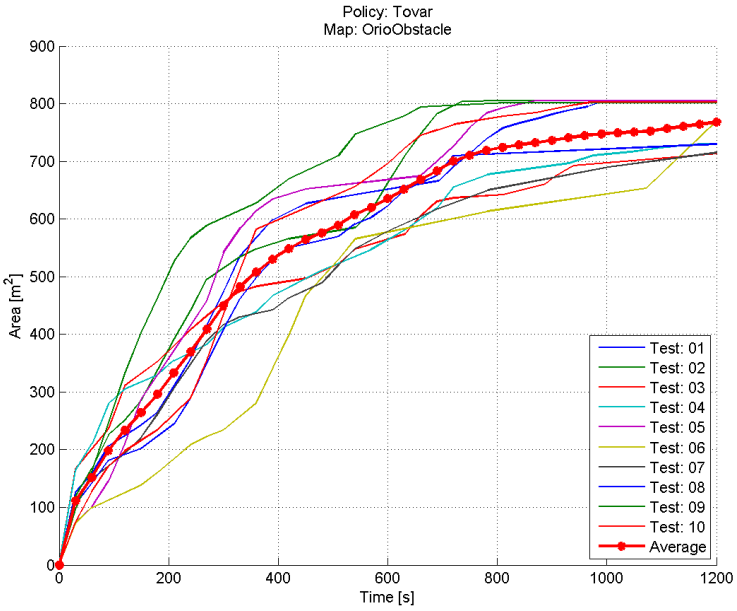

Figura 5.33: Grafico che rappresenta i risultati dei test eseguiti sulla mappa Obstacle con la strategia di Tovar.

Nei grafici seguenti sono state confrontate le medie e le deviazioni standard (Figura 5.34) ottenute elaborando i risultati dei grafici precedenti.

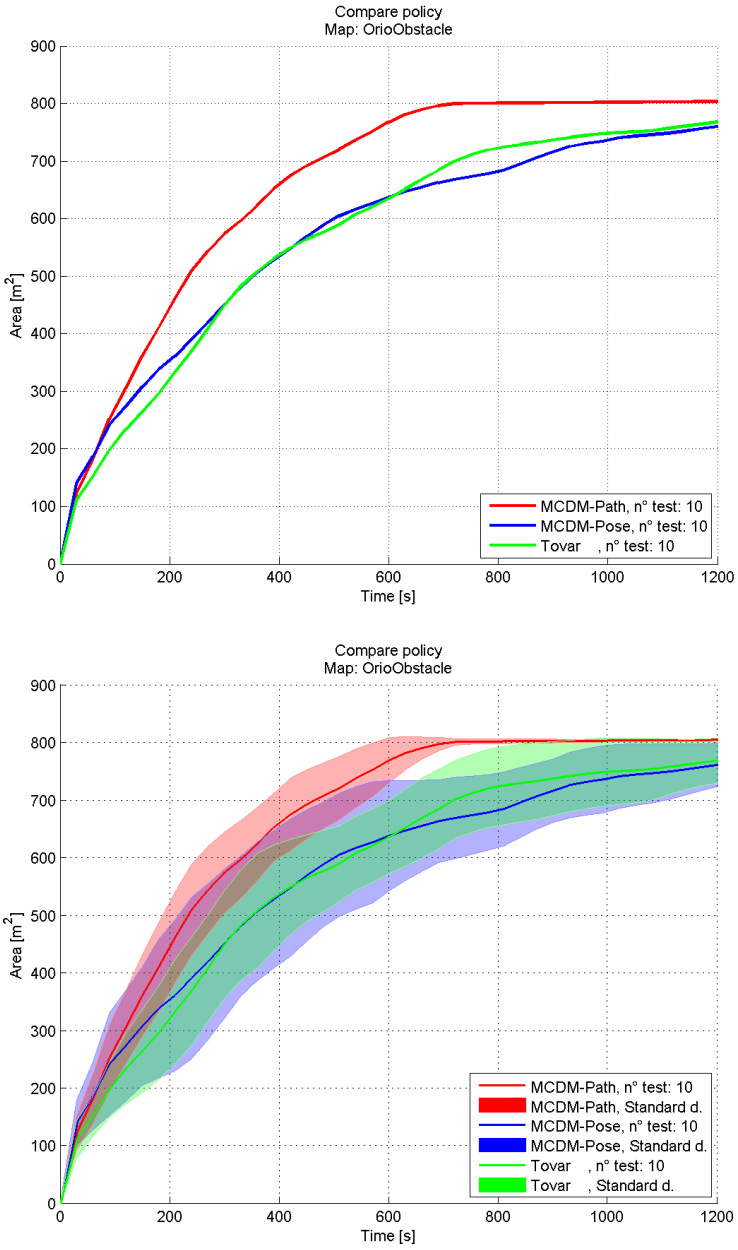

Figura 5.34: In alto è mostrato il confronto tra le medie dei risultati ottenuti con i dieci test sulla mappa Obstacle delle tre strategie implementate, mentre in basso si può notare il confronto tra le deviazioni standard.

Per ottenere dei risultati statisticamente rilevanti, è stato eseguito un test *ANOVA* con soglia di significatività α pari a 0.05. Per mostrare in modo evidente che le differenze nei grafici precedenti non sono solo frutto del caso, è necessario che i p-value risultanti dalle tabelle seguenti siano inferiori al valore α.

Confronto dell'area misurata ad un orizzonte temporale di 720s.

- confronto con la strategia MCDM-Pose
  - i risultati completi sono riportati nella tabella 5.15 mentre nella figura 5.35 sono presentati in forma grafica

Source	SS	df	MS	F	p-value
Columns	91353	1	91353	30.14	**0.00003**
Error	54548	18	3030	-	-
Total	145901	19	-	-	-

Tabella 5.15: Risultati del test ANOVA sull'area misurata della mappa Obstacle, dalle strategie MCDM-Path e MCDM-Pose, ad un orizzonte temporale di 720s.

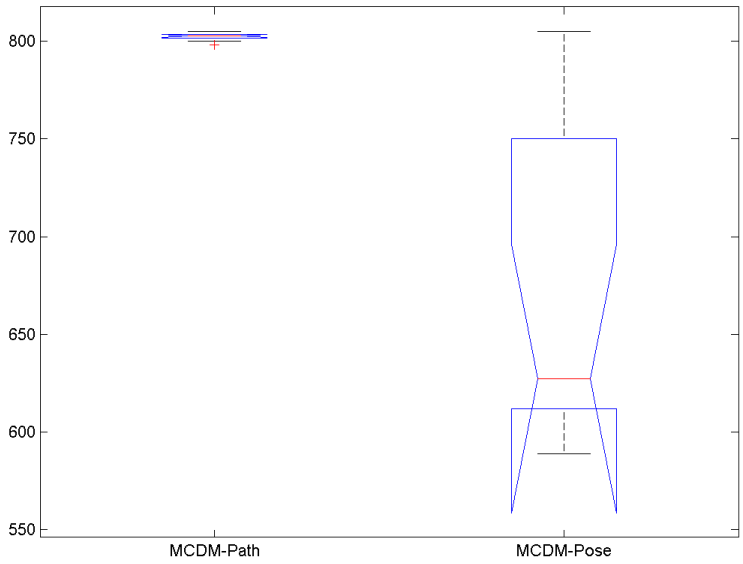

Figura 5.35: Risultati del test ANOVA sull'area misurata della mappa Obstacle, dalle strategie MCDM-Path e MCDM-Pose, ad un orizzonte temporale di 720s.

- confronto con la strategia Tovar
  - la tabella 5.16 e l'immagine 5.36 mostrano i risultati ottenuti con il test *ANOVA*

Source	SS	df	MS	F	p-value
Columns	49830	1	49830	14.28	**0.0014**
Error	62792	18	3488	-	-
Total	112623	19	-	-	-

Tabella 5.16: Risultati del test ANOVA sull'area misurata della mappa Obstacle, dalle strategie MCDM-Path e Tovar, ad un orizzonte temporale di 720s.

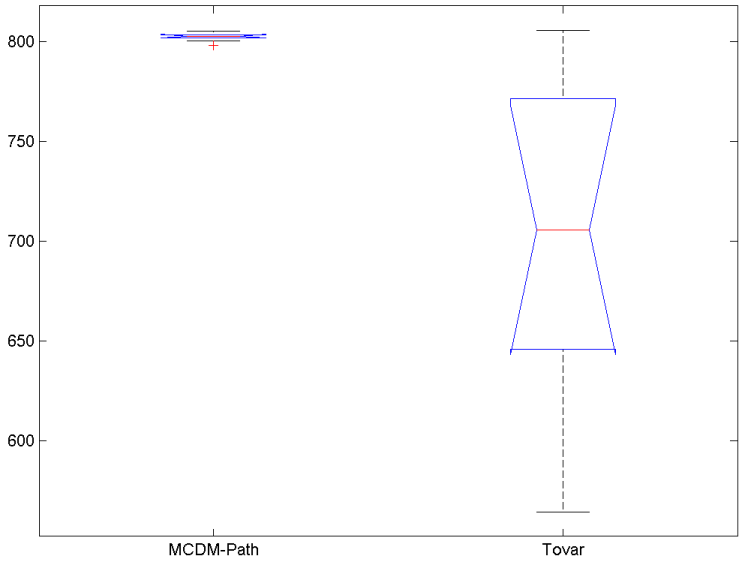

Figura 5.36: Risultati del test ANOVA sull'area misurata della mappa Obstacle, dalle strategie MCDM-Path e Tovar, ad un orizzonte temporale di 720s.

Confronto del tempo impiegato per mappare l'80% dell'area.

- confronto con la strategia MCDM-Pose

  – i risultati ottenuti sono riportati nella tabella 5.17 e nell'immagine 5.37

Source	SS	df	MS	F	p-value
Columns	340605	1	340605	11.04	**0.0038**
Error	555570	18	30865	-	-
Total	896175	19	-	-	-

Tabella 5.17: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa Obstacle, da parte dalle strategie MCDM-Path e MCDM-Pose.

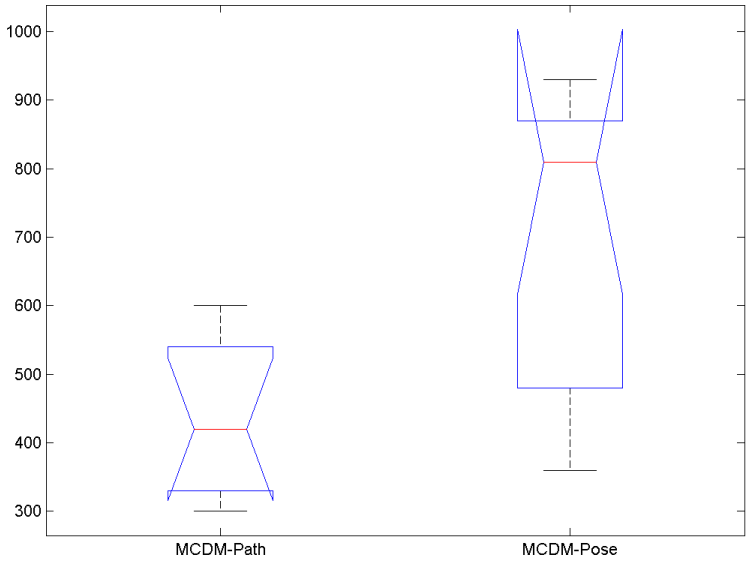

Figura 5.37: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa Obstacle, da parte dalle strategie MCDM-Path e MCDM-Pose.

- confronto con la strategia Tovar
  - la tabella 5.18 e l'immagine 5.38 raffigurano i risultati ottenuti confrontando la strategia MCDM-Path con la strategia Tovar

Source	SS	df	MS	F	p-value
Columns	253125	1	253125	8.96	**0.0078**
Error	508770	18	28265	-	-
Total	761895	19	-	-	-

Tabella 5.18: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa Obstacle, da parte dalle strategie MCDM-Path e Tovar.

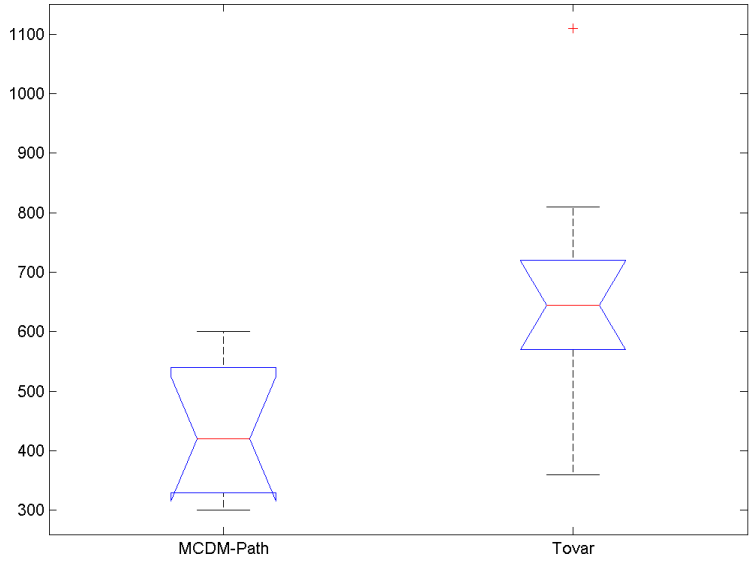

Figura 5.38: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa Obstacle, da parte dalle strategie MCDM-Path e Tovar.

I risultati ottenuti dai test su questa mappa evidenziano come l'utilizzo della strategia MCDM-Path permetta di ottenere dei vantaggi tangibili aumentando la complessità dell'ambiente. Questa mappa, ricca di ostacoli che impediscono una percezione chiara e immediata dello spazio, rende l'utilizzo dell'information gain ottenuta lungo l'intero cammino, un fattore che avvantaggia notevolmente la strategia da noi implementata. Una scelta accurata del percorso da seguire permette, infatti, l'ottimizzazione del tempo a disposizione e la riduzione della distanza percorsa.

5.3.3.3 Mix

La mappa Mix (Figura 5.39), estratta dall'ambiente Orio, è caratterizzata sia da ampi spazi aperti con una scarsa presenza di ostacoli, nei quali il movimento e la percezione del robot sono più agevoli, sia da stanze di dimensioni ridotte che rendono il movimento e la percezione dell'agente più complicati.

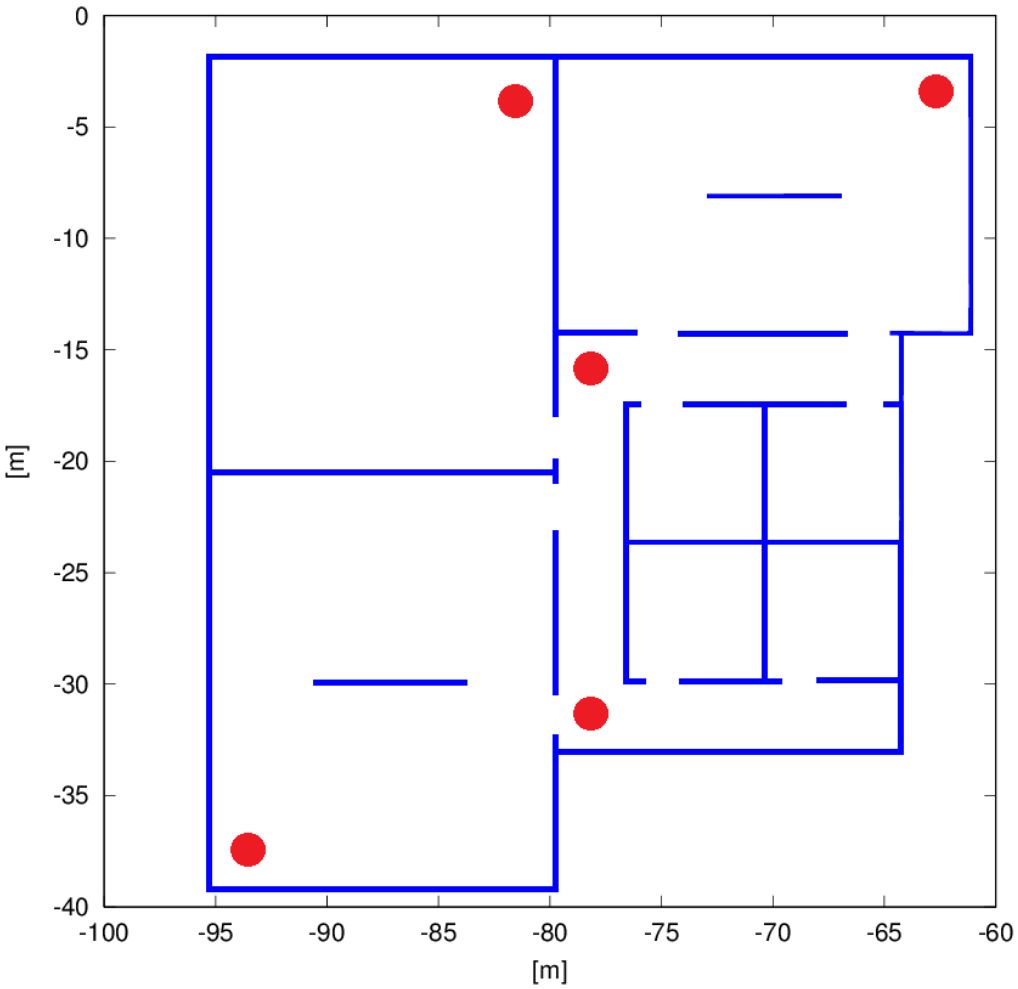

Figura 5.39: Rappresentazione dell'ambiente Orio Mix di area 1107 m^2.

Le figure seguenti mostrano i risultati delle cinque prove eseguite nella mappa Mix, utilizzando le tre strategie implementate: MCDM-Path (Figura 5.40), MCDM-Pose (Figura 5.41) e Tovar (Figura 5.42). La linea rossa punteggiata mostra la media dei cinque test effettuati.

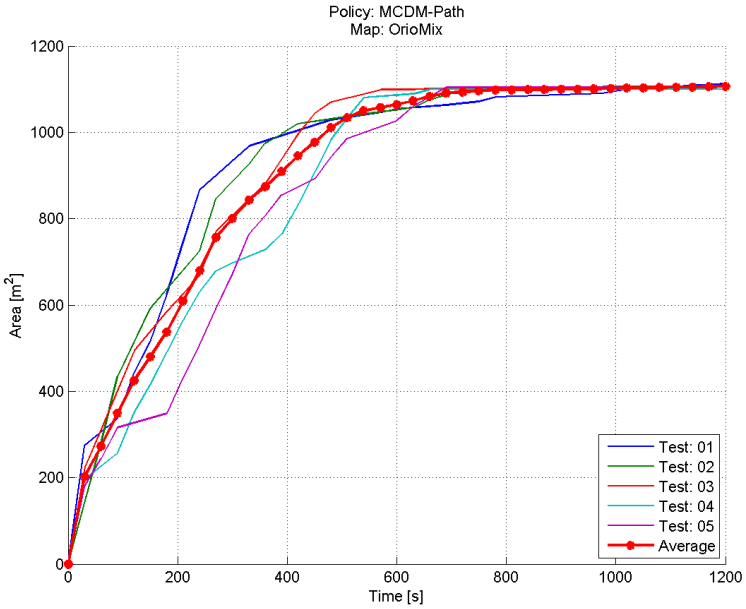

Figura 5.40: Grafico che rappresenta i risultati dei test eseguiti sulla mappa Mix con la strategia MCDM-Path.

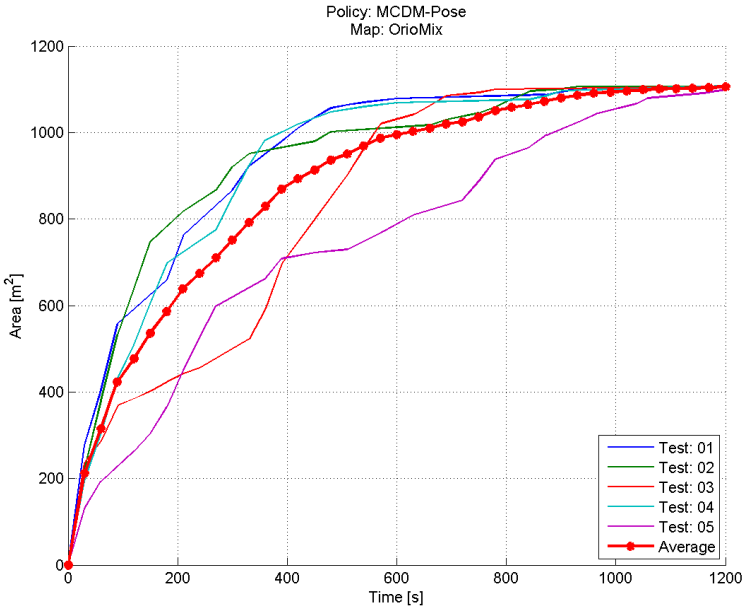

Figura 5.41: Grafico che rappresenta i risultati dei test eseguiti sulla mappa Mix con la strategia MCDM-Pose.

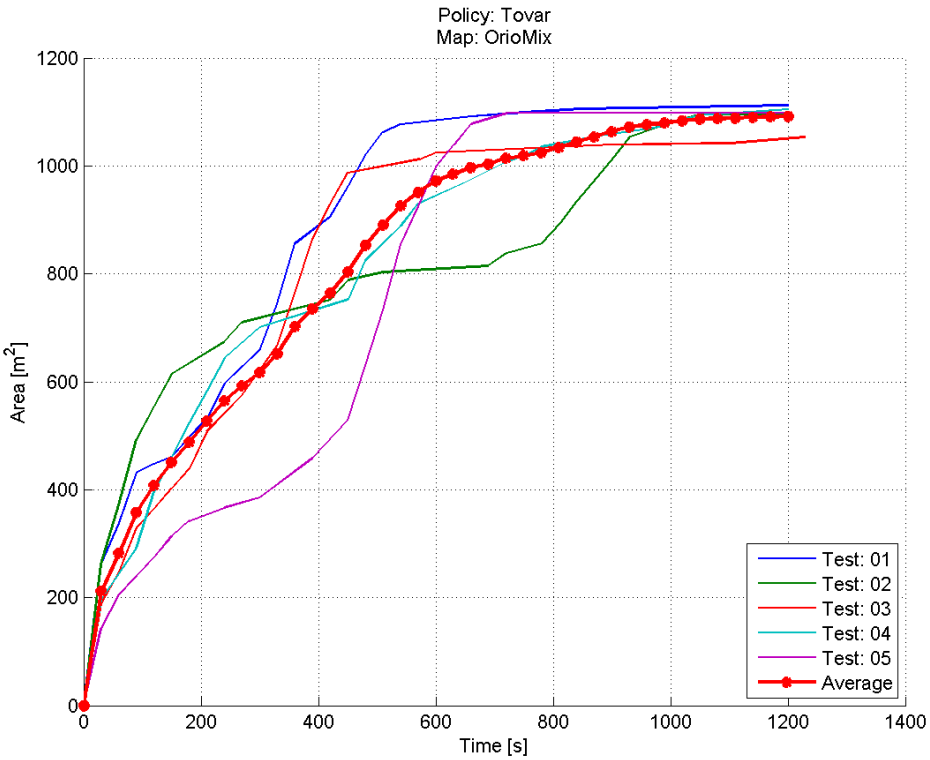

Figura 5.42: Grafico che rappresenta i risultati dei test eseguiti sulla mappa Mix con la strategia di Tovar et al..

Nei grafici seguenti sono state confrontate le medie e le deviazioni standard (Figura 5.43) ottenute elaborando i risultati dei grafici precedenti.

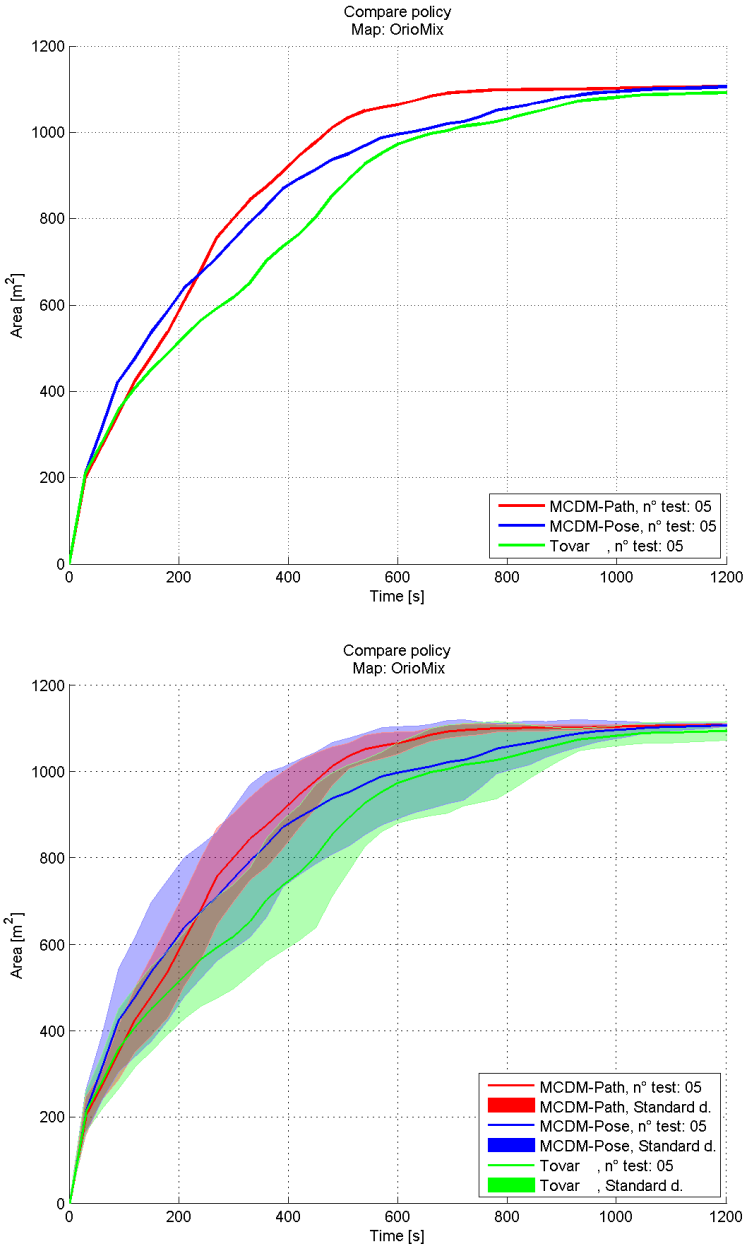

Figura 5.43: In alto è mostrato il confronto tra le medie dei risultati ottenuti con i cinque test sulla mappa Mix delle tre strategie implementate, mentre in basso si può notare il confronto tra le deviazioni standard.

La sola analisi dei grafici precedenti non è sufficiente. Per un confronto più significativo tra le strategie adottate è stato eseguito un test *ANOVA*. Per evidenziare delle differenze significative nelle prestazioni delle strategie utilizzate, è necessario che i p-value risultanti dalle tabelle seguenti siano inferiori a 0.05.

Confronto dell'area misurata ad un orizzonte temporale di 780s.

- confronto con la strategia MCDM-Pose
  - i risultati completi sono riportati nella tabella 5.19 mentre nell'immagine 5.44 sono presentati in forma grafica

Source	SS	df	MS	F	p-value
Columns	6402	1	6402	2.09	**0.1859**
Error	24457	8	3057	-	-
Total	30860	9	-	-	-

Tabella 5.19: Risultati del test ANOVA sull'area misurata della mappa Mix, dalle strategie MCDM-Path e MCDM-Pose, ad un orizzonte temporale di 780s.

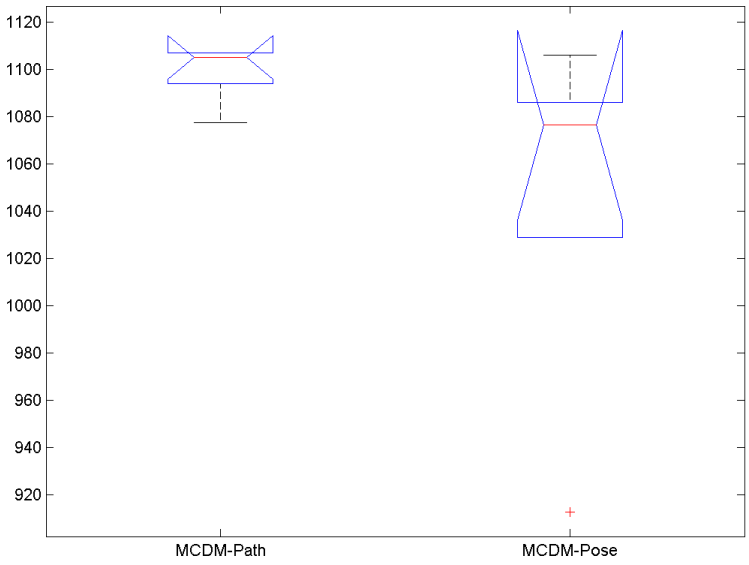

Figura 5.44: Risultati del test ANOVA sull'area misurata della mappa Mix, dalle strategie MCDM-Path e MCDM-Pose, ad un orizzonte temporale di 780s.

- confronto con la strategia Tovar
  - i risultati ottenuti dal confronto di MCDM-Path con la strategia Tovar, sono riportati nella tabella 5.20 e nell'immagine 5.45

Source	SS	df	MS	F	p-value
Columns	15994	1	15994	2.52	**0.1511**
Error	50777	8	6347	-	-
Total	66772	9	-	-	-

Tabella 5.20: Risultati del test ANOVA sull'area misurata della mappa Mix, dalle strategie MCDM-Path e Tovar, ad un orizzonte temporale di 780s.

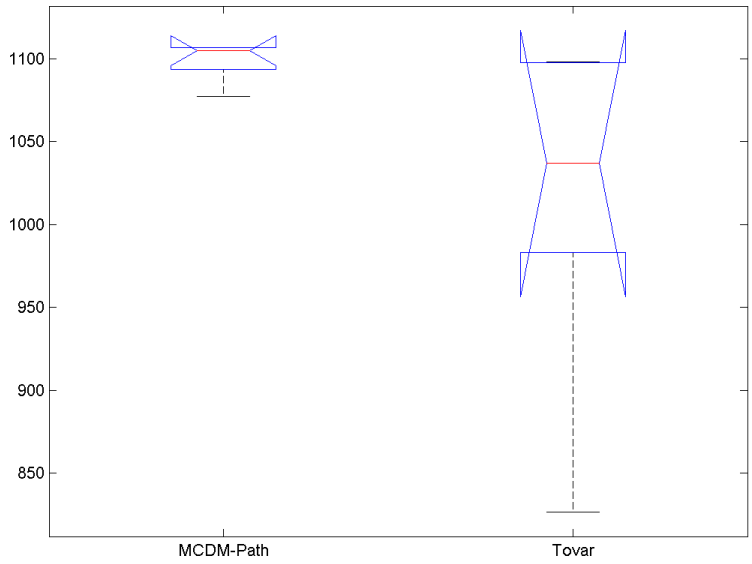

Figura 5.45: Risultati del test ANOVA sull'area misurata della mappa Mix, dalle strategie MCDM-Path e Tovar, ad un orizzonte temporale di 780s.

Confronto del tempo impiegato per mappare l'80% dell'area.

- confronto con la strategia MCDM-Pose
  - La tabella 5.21 e la figura 5.46 mostrano i risultati del confronto con MCDM-Pose

Source	SS	df	MS	F	p-value
Columns	2025	1	2025	9	**0.0955**
Error	450	2	225	-	-
Total	2475	3	-	-	-

Tabella 5.21: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa Mix, da parte dalle strategie MCDM-Path e MCDM-Pose.

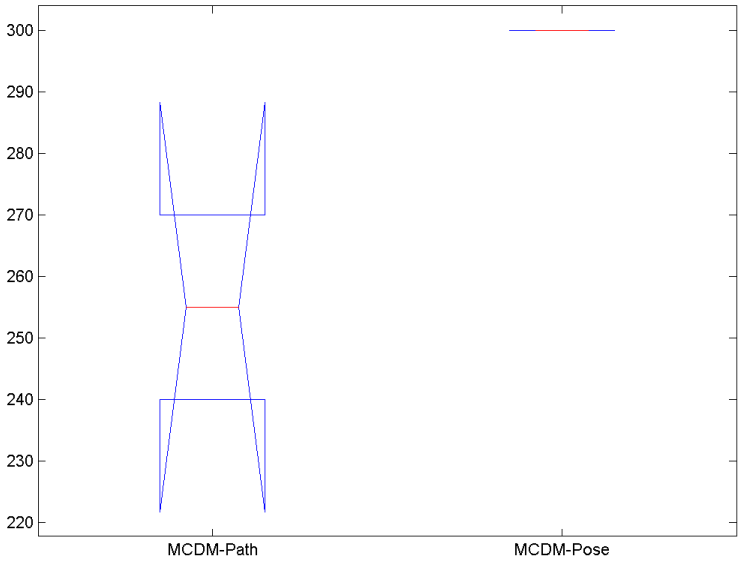

Figura 5.46: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa Mix, da parte dalle strategie MCDM-Path e MCDM-Pose.

- confronto con la strategia Tovar
  - i risultati completi del test *ANOVA*, sono riportati nella tabella 5.22 e nell'immagine 5.47

Source	**SS**	**df**	**MS**	**F**	**p-value**
Columns	129600	1	129600	3.39	**0.207**
Error	76500	2	38250	-	-
Total	206100	3	-	-	-

Tabella 5.22: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa Mix, da parte dalle strategie MCDM-Path e Tovar.

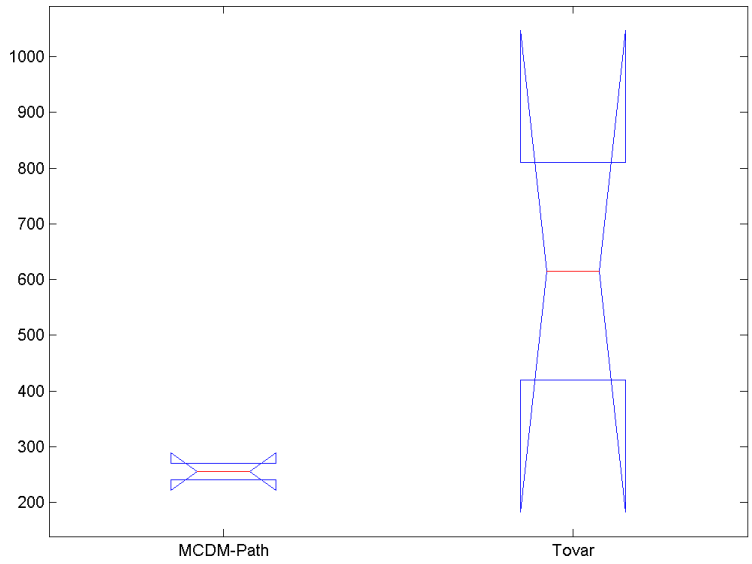

Figura 5.47: Risultati del test ANOVA sul tempo impiegato per mappare l'80% dell'area della mappa Mix, da parte dalle strategie MCDM-Path e Tovar.

Osservando i risultati dei test effettuati, non è possibile trovare una strategia vincente tra quelle da noi valutate. L'analisi della varianza mostra un leggero vantaggio delle prestazioni ottenute con la strategia MCDM-Path, non abbastanza marcato, però, da poter dimostrare in modo statisticamente rilevante la superiorità di una strategia rispetto alle altre.

5.3.4 Mappa VascheLibraryFloor1 (Repository Radish)

Da questa mappa (Figura 5.4), proveniente dal lavoro originale su cui si basa questa tesi, è stato estratto un ambiente, con caratteristiche simili alla mappa Orio Obstacle, adatto all'esplorazione con singolo agente.

5.3.4.1 Radish

La mappa Radish (Figura 5.48), estratta dall'ambiente VascheLibrary-Floor1, è caratterizzata da stanze di medie e grandi dimensioni con la presenza di ostacoli che ne impediscono la visione completa al momento dell'ingresso nelle stesse.

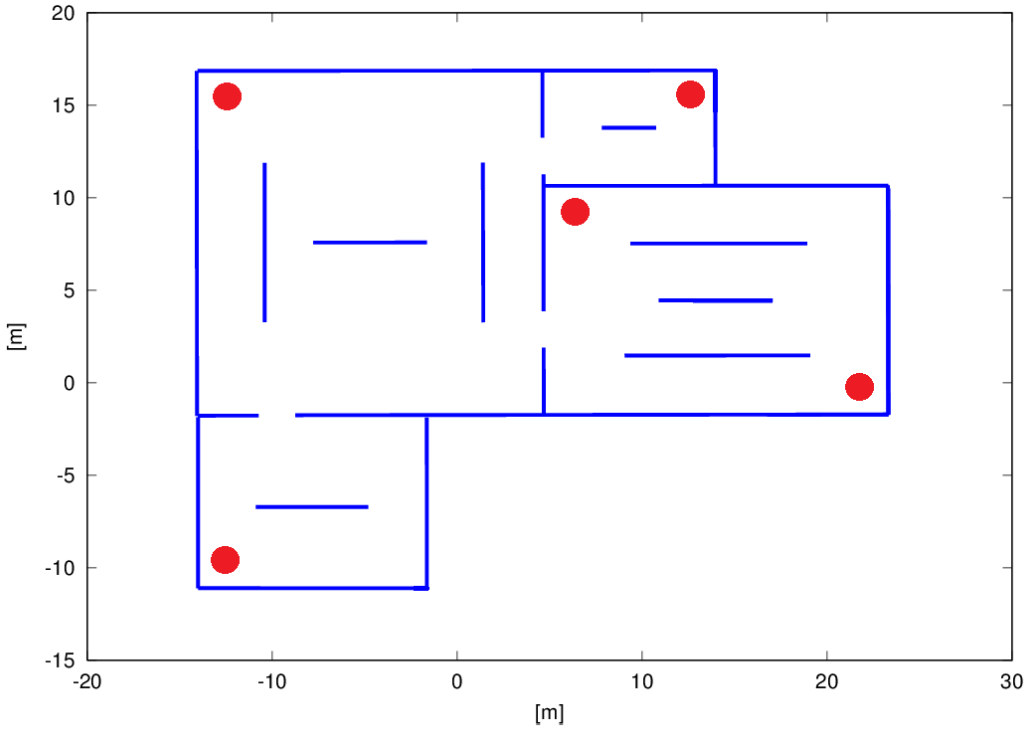

Figura 5.48: Rappresentazione dell'ambiente Radish di area 755.5 m^2.

Nelle immagini seguenti vengono presentati i risultati delle cinque prove eseguite con le tre strategie implementate: MCDM-Path (Figura 5.49), MCDM-Pose (Figura 5.50) e Tovar (Figura 5.51).

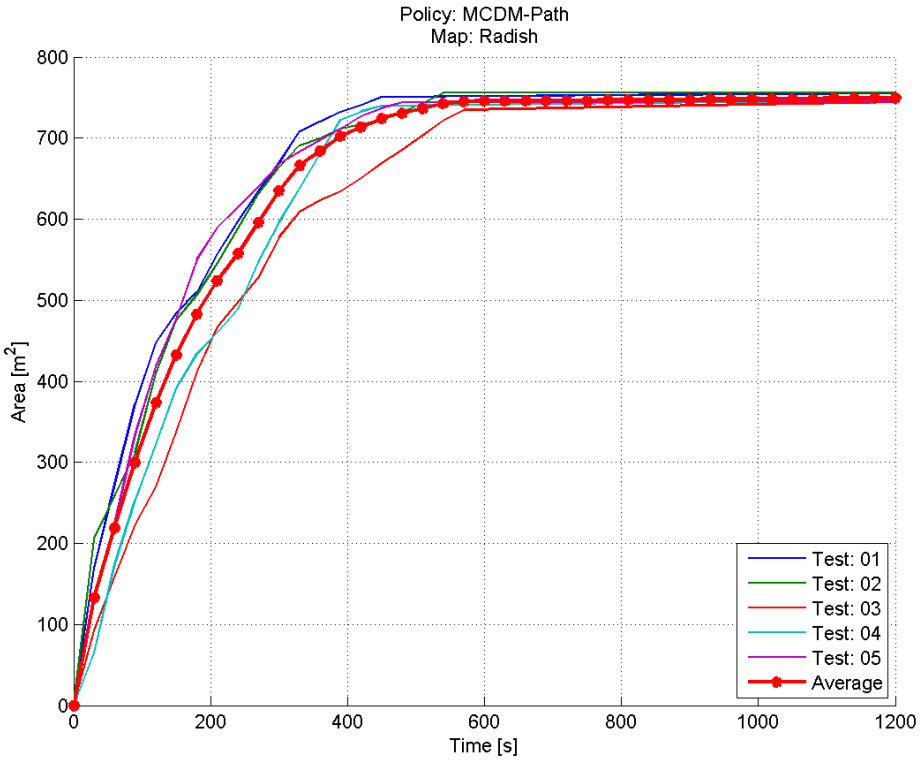

Figura 5.49: Grafico che rappresenta i risultati dei test eseguiti sulla mappa Radish con la strategia MCDM-Path.

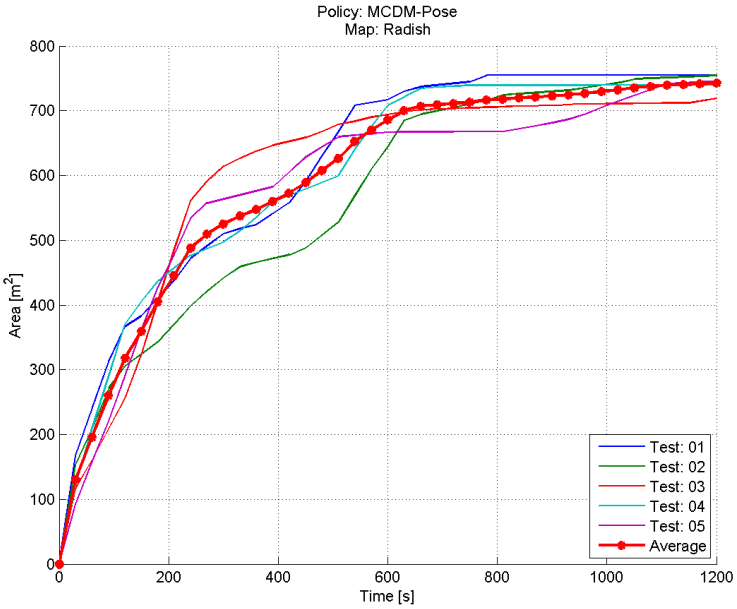

Figura 5.50: Grafico che rappresenta i risultati dei test eseguiti sulla mappa Radish con la strategia MCDM-Pose.

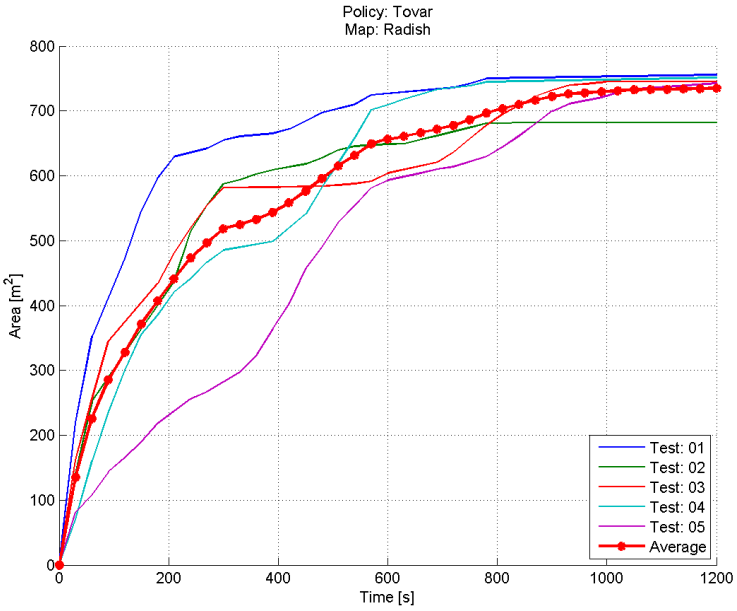

Figura 5.51: Grafico che rappresenta i risultati dei test eseguiti sulla mappa Radish con la strategia di Tovar.

Le medie e le deviazioni standard ottenute confrontando i risultati dei test effettuati sulla mappa Radish, sono mostrate nei grafici seguenti (Figura 5.52)

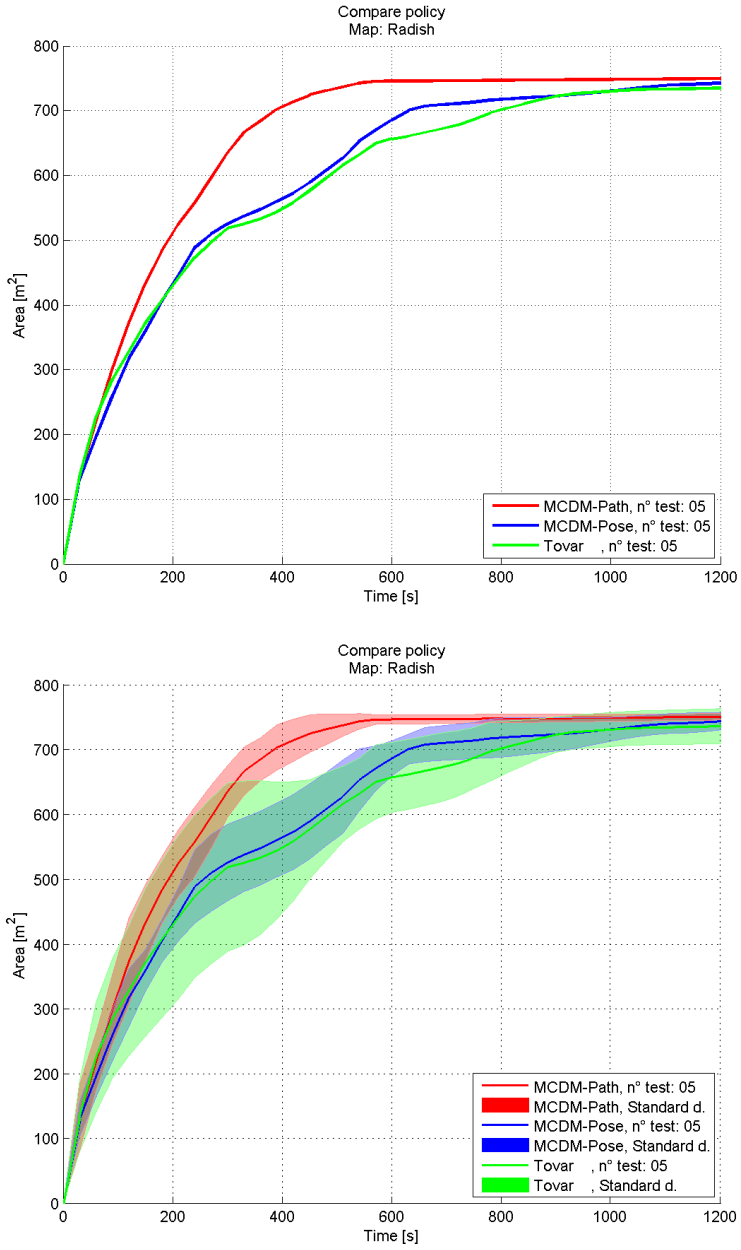

Figura 5.52: In alto è mostrato il confronto tra le medie dei risultati ottenuti con i cinque test sulla mappa Radish delle tre strategie implementate, mentre in basso si può notare il confronto tra le deviazioni standard.

I successivi test *ANOVA* sono stati utilizzati per mostrare se esiste una differenza evidente nelle prestazioni delle strategie utilizzate, come sembrerebbero mostrare i grafici precedenti.

Confronto dell'area misurata ad un orizzonte temporale di 600s.

- confronto con la strategia MCDM-Pose

– i risultati ottenuti sono riportati nella tabella 5.23 e nella figura 5.53

Source	SS	df	MS	F	p-value
Columns	7333	1	7333	19.50	**0.0022**
Error	3008	8	376	-	-
Total	10341	9	-	-	-

Tabella 5.23: Risultati del test ANOVA sull'area misurata della mappa Radish, dalle strategie MCDM-Path e MCDM-Pose, ad un orizzonte temporale di 600s.

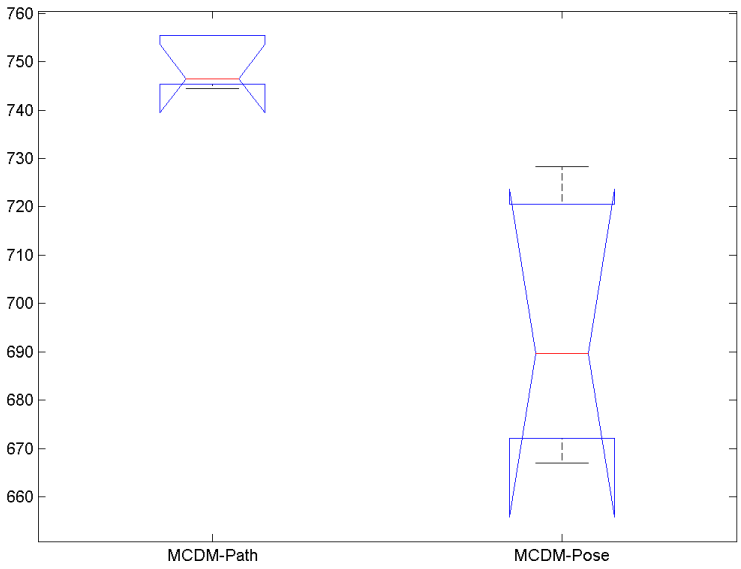

Figura 5.53: Risultati del test ANOVA sull'area misurata della mappa Radish, dalle strategie MCDM-Path e MCDM-Pose, ad un orizzonte temporale di 600s.

- confronto con la strategia Tovar

  – i risultati del test *ANOVA* tra la strategia MCDM-Path e quella Tovar, sono riportati nella tabella 5.24 e nell'immagine 5.54

Source	SS	df	MS	F	p-value
Columns	21684	1	21684	10.99	**0.0106**
Error	15792	8	1974	-	-
Total	37476	9	-	-	-

Tabella 5.24: Risultati del test ANOVA sull'area misurata della mappa Radish, dalle strategie MCDM-Path e Tovar, ad un orizzonte temporale di 600s.

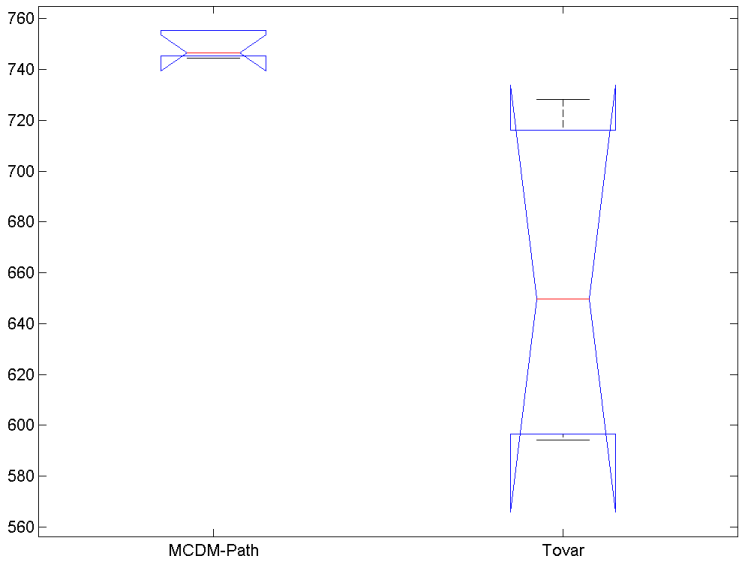

Figura 5.54: Risultati del test ANOVA sull'area misurata della mappa Radish, dalle strategie MCDM-Path e Tovar., ad un orizzonte temporale di 600s.

Confronto del tempo impiegato per mappare il 90% dell'area.

- confronto con la strategia MCDM-Pose

  – la tabella 5.25 mostra i risultati ottenuti dall'analisi delle varianze in forma testuale, mentre nell'immagine 5.55 sono presentati in forma grafica

Source	SS	df	MS	F	p-value
Columns	190440	1	190440	10.85	**0.011**
Error	140400	8	17550	-	-
Total	330840	9	-	-	-

Tabella 5.25: Risultati del test ANOVA sul tempo impiegato per mappare il 90% dell'area della mappa Radish, da parte dalle strategie MCDM-Path e MCDM-Pose.

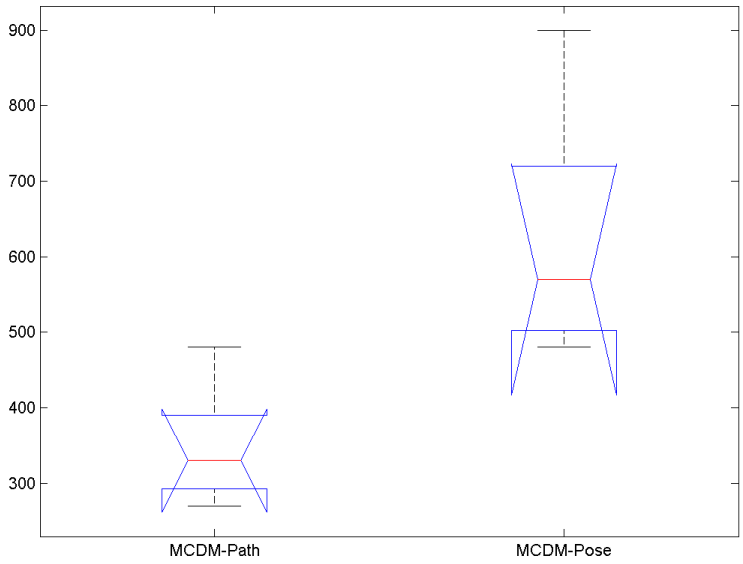

Figura 5.55: Risultati del test ANOVA sul tempo impiegato per mappare il 90% dell'area della mappa Radish, da parte dalle strategie MCDM-Path e MCDM-Pose.

- confronto con la strategia Tovar
  - i risultati completi sono riportati nella tabella 5.26 mentre nella figura 5.56 sono presentati in forma grafica

Source	SS	df	MS	F	p-value
Columns	272250	1	272250	15.28	**0.0045**
Error	142560	8	17820	-	-
Total	414810	9	-	-	-

Tabella 5.26: Risultati del test ANOVA sul tempo impiegato per mappare il 90% dell'area della mappa Radish, da parte dalle strategie MCDM-Path e Tovar.

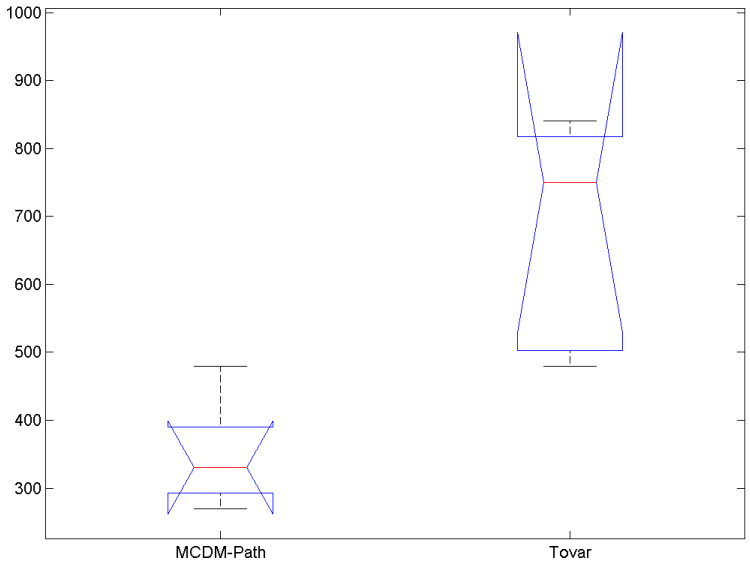

Figura 5.56: Risultati del test ANOVA sul tempo impiegato per mappare il 90% dell'area della mappa Radish, da parte dalle strategie MCDM-Path e Tovar.

I test eseguiti su questa mappa, con caratteristiche simili all'ambiente Obstacle (Sezione 5.3.3.2), mostrano le performance significativamente maggiori ottenute utilizzando la strategia MCDM-Path. Una valutazione del percorso da seguire permette, infatti, l'ottimizzazione del tempo a disposizione e la riduzione della distanza percorsa (come mostrato nel confronto effettuato sui test dell'ambiente Radish (Figura 5.57)), su mappe che presentano ostacoli che riducono la capacità del robot di percepire l'ambiente circostante. Uno studio approfondito attraverso il test *ANOVA*, mostra p-value di molto inferiori al valore α fissato come riferimento, confermando le conclusioni relative alla significatività statistica dei risultati a cui siamo giunti in precedenza osservando i grafici.

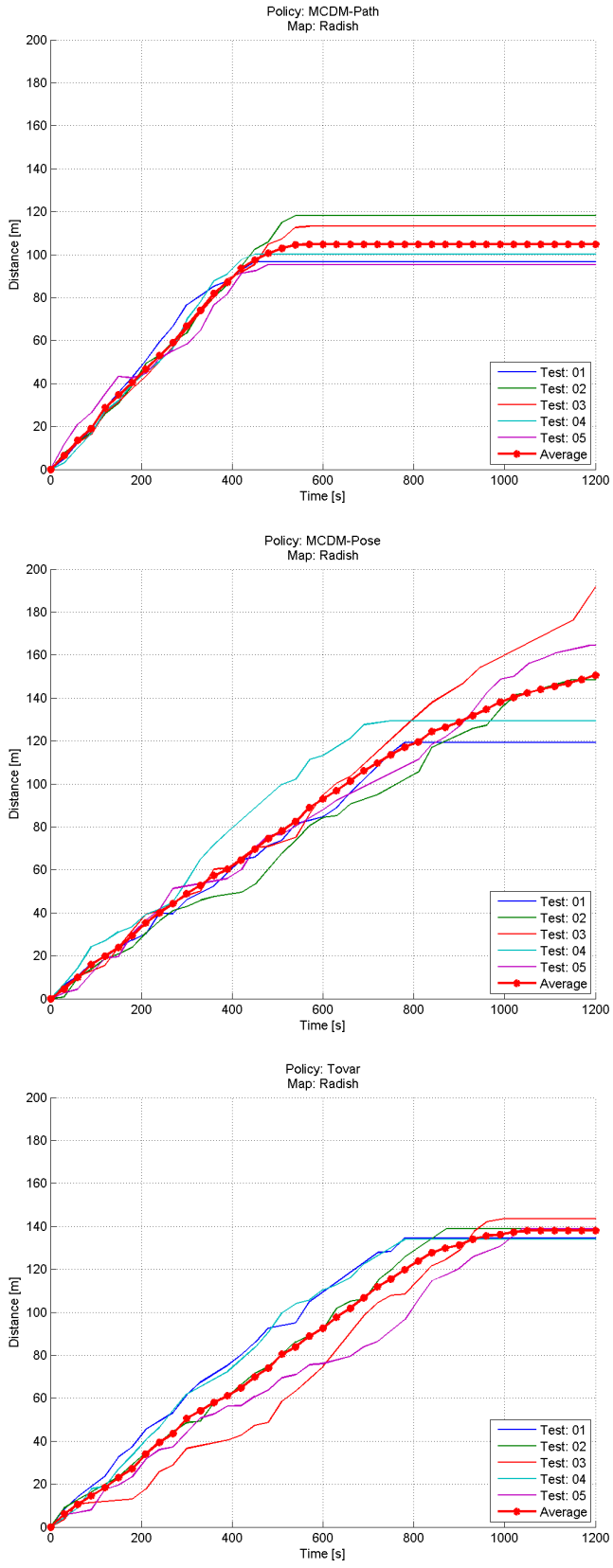

Figura 5.57: Grafici che mostrano la distanza percorsa dal robot durante l'esplorazione dell'ambiente Radish con le tre strategie implementate.

5.3.5 Confronto delle valutazioni dei path

In questa sezione sono presentati i risultati ottenuti mettendo a confronto le valutazioni, assegnate dalle tre strategie prese in esame, ad alcuni path. Questo è stato possibile grazie all'implementazione di una porzione di codice da noi modificato ad hoc, che ha permesso la scrittura nei file di log dei valori risultanti dalla valutazione dei tre path per tutte e tre le strategie implementate.

Il primo confronto (Figura 5.58), è stato fatto sull'ambiente Radish, caratterizzato dalla presenza di ostacoli che impediscono la vista di un'intera stanza con una sola percezione. Questo test in particolare si riferisce al calcolo e alla valutazione di percorsi necessari per raggiungere una frontiera posizionata in una altra stanza.

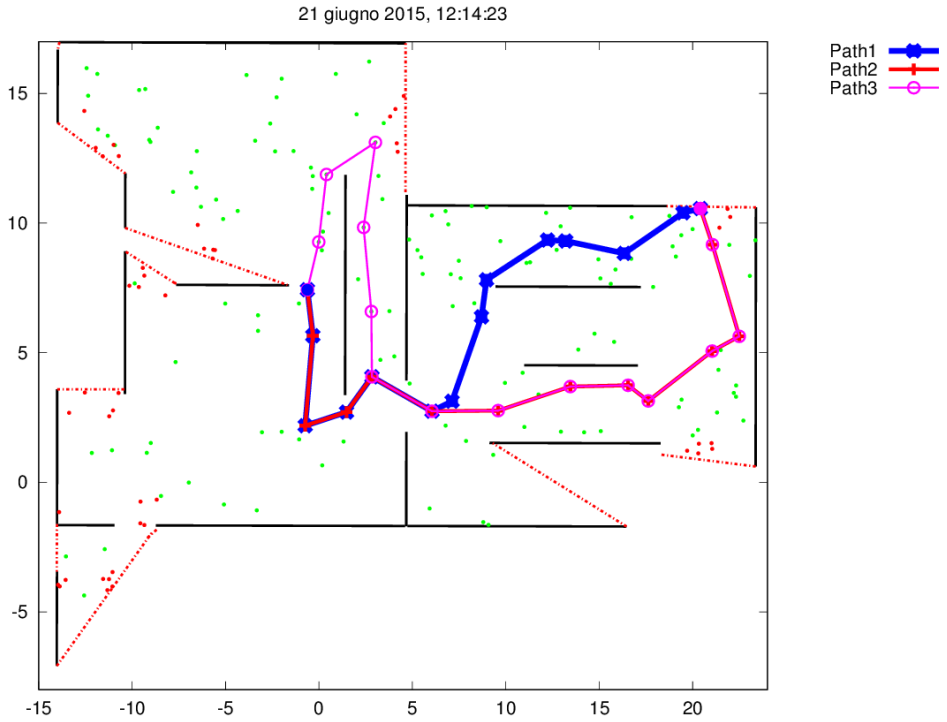

Figura 5.58: Area parziale della mappa Radish percepita dal robot dove si possono notare le frontiere (identificate dalle linea rosse tratteggiate) e i tre cammini che il robot può percorrere dalla posizione iniziale (punto blu) per arrivare alla posizione di destinazione (punto viola). Le linee nere identificano muri ed ostacoli. I tre path sono stati identificati per colore: blu, rosso e viola. Le stesse convenzioni sono state usate per le figure dei confronti che seguono.

Nella tabella 5.27, sono riportati i valori che identificano le valutazioni assegnate ai tre path creati, per ognuna delle tre strategie utilizzate.

Strategie	**Frontiera di riferimento**	**Valori {path1, path2, path3}**
Tovar	{21.002405,10.630288}	{11.910965, 11.917484, **36.028634**}
MCDM-Pose	{21.002405,10.630288}	{**0.546977**, 0.509029, 0.446922}
MCDM-Path	{21.002405,10.630288}	{0.337594, 0.397988, **0.515225**}

Tabella 5.27: Valori assegnati ai tre path per ognuna delle tre strategie implementate, con evidenziati i valori migliori. La frontiera viene identificata con le coordinate (x, y) del centroide .

Osservando i risultati precedenti (Figura 5.58) (Tabella 5.27), possiamo osservare che il path scelto dalla strategia Tovar è il percorso viola, quello scelto dalla MCDM-Pose è il blu, mentre quello preferito della strategia MCDM-Path è nuovamente il cammino di colore viola.

Altri confronti sono stati fatti sui risultati ottenuti dai test sull'ambiente VMAC2011 (Figura 5.59), che, come già detto in precedenza, presenta stanze di medie dimensioni con una scarsa presenza di ostacoli, unite da corridoi facilmente percorribili dal robot. I due test presentati mostrano entrambi una situazione in cui il robot deve raggiungere una stanza situata molto lontano dalla sua posizione attuale. Questo ci ha permesso di ottenere dei cammini di lunghezza elevata aumentando così la probabilità che i percorsi variassero in modo significativo tra loro, nonostante la scarsa presenza di ostacoli.

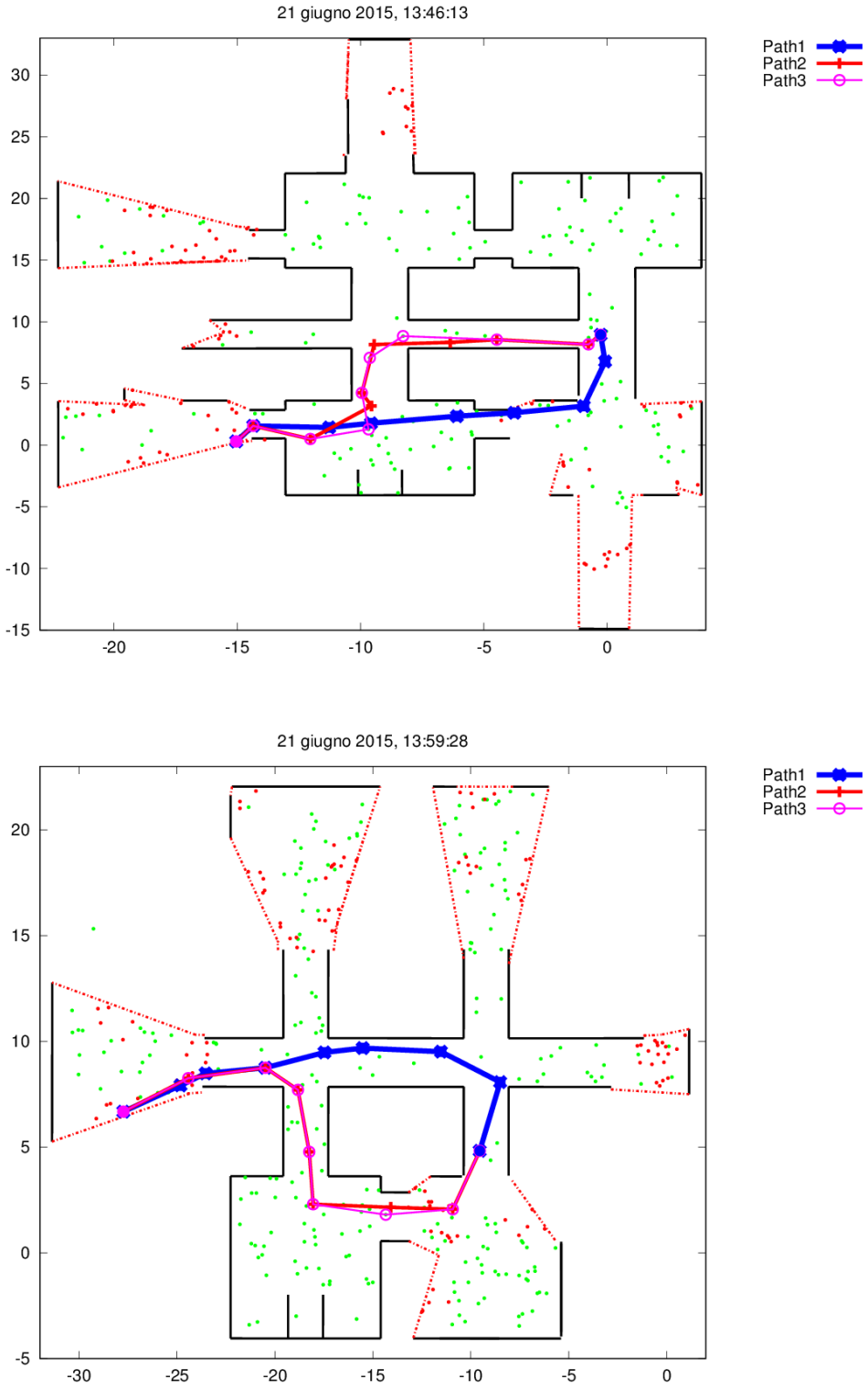

Figura 5.59: Area parziale della mappa VMAC2011, sulla quale sono stati eseguiti due confronti della valutazione dei path.

Nella tabella 5.28, sono riportati i valori che identificano le valutazioni assegnate ai tre path creati, per ognuna delle tre strategie utilizzate.

Strategie	Frontiera di riferimento	Valori {path1, path2, path3}
Tovar	{-14.679489,0.292570}	{0.643330, **0.068319**, 0.062296}
MCDM-Pose	{-14.679489,0.292570}	{**0.365956**, 0.334911, 0.321875}
MCDM-Path	{-14.679489,0.292570}	{**0.630128**, 0.513895, 0.507705}

Strategie	Frontiera di riferimento	Valori {path1, path2, path3}
Tovar	{-14.679489,0.292570}	{0.643330, **0.068319**, 0.062296}
MCDM-Pose	{-14.679489,0.292570}	{**0.365956**, 0.334911, 0.321875}
MCDM-Path	{-14.679489,0.292570}	{**0.630128**, 0.513895, 0.507705}

Tabella 5.28: Valori assegnati ai tre path per ognuna delle tre strategie, nei due confronti effettuati, con evidenziati i valori migliori. Le frontiere vengono identificate con le coordinate (x, y) dei rispettivi centroidi.

Osservando i risultati (Figura 5.59) (Tabella 5.28), possiamo notare che, sia nel primo che nel secondo confronto, il path scelto dalla strategia Tovar è il cammino di colore rosso, quello scelto dalla MCDM-Pose è il blu, mentre quello preferito della strategia MCDM-Path è nuovamente di colore blu.

Dai risultati ottenuti con il confronto della valutazione dei path, possiamo dedurre che la strategia MCDM-Pose sembra preferire i path più brevi, MCDM-Path prediligere i cammini con information gain più elevata lungo tutti i percorsi, mentre la strategia Tovar pare preferire i path che passano più vicino alle frontiere.

Quest'ultima considerazione, potrebbe giustificare le performance migliori di MCDM-Path rispetto alla strategia Tovar. Per come sono stati equipaggiati i robot durante le nostre simulazioni, infatti, l'acquisizione delle informazioni sull'ambiente non necessita di passare vicino alle frontiere, come fatto dalla strategia Tovar, ma sembra che sia sufficiente intercettarne molte lungo il percorso, come ottenuto con la strategia MCDM-Path.

Capitolo 6

Conclusioni

6.1 Considerazioni finali

Nel campo della robotica mobile, la capacità di esplorare ambienti sconosciuti in modo autonomo è considerata come una delle tematiche che suscita maggior interesse nella comunità scientifica. Il lavoro presentato in questa tesi si colloca in questo campo e, più in particolare, nei contesti denominati Urban Search And Rescue (USAR) in ambienti interni. In queste applicazioni vengono utilizzati uno o più robot per la ricerca di eventuali vittime in edifici coinvolti in incidenti o disastri naturali. L'assenza o la scarsità di informazioni disponibili a priori sull'ambiente in cui si opera permette di semplificare la soluzione del problema, concentrandosi esclusivamente sul massimizzare l'area esplorata nel minor tempo possibile. Questo comporta una scelta accurata della strategia di esplorazione, che deve adattarsi all'ambiente in cui si trova l'agente autonomo e sfruttare a pieno il poco tempo a disposizione per portare a termine il proprio compito.

In questa tesi abbiamo proposto una strategia di esplorazione, basata sul framework MCDM, che estende la classica valutazione della posizione di destinazione con la valutazione del percorso seguito per raggiungerla. Il nostro lavoro si è concentrato sull'applicare questa strategia al codice già esistente implementato dal team PoAReT, vincitore della Virtual Robot Competition nell'ambito di RoboCup2012. Per fare questo è stato necessario modificare principalmente il modulo Path Planner, in modo da ottenere un insieme di cammini percorribili dal robot, verso ciascuna delle possibili destinazioni. E' stato inoltre fondamentale cambiare la politica di valutazione, responsabile della scelta del prossimo punto da raggiungere, in modo che considerasse le in-

formazioni aggiuntive potenzialmente ottenibili dal robot durante il movimento verso la destinazione scelta.

Per la valutazione delle prestazioni, sono stati condotti esperimenti con il simulatore USARSim su mappe con caratteristiche diverse tra loro, confrontando la nostra implementazione (chiamata MCDM-Path) con i risultati di due strategie di valutazione alternative: MCDM-Pose (che valuta solo la bontà delle posizioni di destinazione) e Tovar (che valuta anche la bontà dei percorsi per raggiungere le posizioni). I confronti sono stati eseguiti comparando l'area media mappata nel tempo usando le tre strategie prese in considerazione e dei test ANOVA sono stati eseguiti per rilevare la significatività statistica dei campioni raccolti.

I risultati ottenuti mostrano, almeno nell'ambito dei test eseguiti, un incremento delle prestazioni in determinati ambienti, dovuto all'utilizzo di una strategia di esplorazione che valuta l'intero percorso seguito dall'agente autonomo. Una tale strategia può portare benefici rispetto ad una alternativa che si concentra esclusivamente sulla scelta della destinazione, soprattutto in ambienti ricchi di ostacoli che impediscono al robot di acquisire informazioni su grandi spazi della mappa con poche percezioni. I metodi basati sulla valutazione dei percorsi, inoltre, presentano un comportamento più stabile, ottenendo dei risultati con una varianza più bassa. Un ulteriore miglioramento rispetto alla strategia MCDM-Pose, è rappresentato dalla riduzione della distanza percorsa per mappare completamente l'area assegnata. Questo è dovuto alla capacità di scegliere con più accuratezza il percorso da seguire, scartando i percorsi che possono sembrare più brevi, ma guadagnando a lungo termine sulla distanza percorsa complessivamente, migliorando così l'efficienza.

Un ulteriore confronto può essere fatto sulle due strategie di valutazione dei path implementate: MCDM-Path e Tovar. I risultati ottenuti mostrano come l'utilizzo del framework MCDM garantisca una maggiore flessibilità nella scelta dei criteri da utilizzare, rispetto ad una soluzione ad hoc come quella di Tovar, ottenendo complessivamente delle prestazioni migliori.

Un'ultima considerazione, ma non di minore importanza, può essere fatta osservando i risultati ottenuti dal confronto qualitativo dei path scelti. Possiamo notare che la strategia MCDM-Pose preferisce i path più brevi, la MCDM-Path predilige i cammini con information gain più elevata lungo tutto il percorso, mentre la strategia Tovar preferisce i path che passano più vicino alle frontiere.

Quest'ultima osservazione sembra giustificare le performance migliori di

MCDM-Path rispetto alla strategia Tovar. Nelle simulazioni da noi effettuate, infatti, per l'acquisizione delle informazioni sull'ambiente non si ha la necessità di passare eccessivamente vicino alle frontiere, ma basta intercettarne molte lungo il percorso.

6.2 Sviluppi futuri

Nonostante i risultati ottenuti sembrino dimostrare che l'algoritmo di esplorazione dai noi implementato produca un incremento delle prestazioni in alcuni degli ambienti in cui abbiamo eseguito gli esperimenti, il lavoro svolto presenta interessanti possibilità di miglioramento e di innovazione.

Una prima proposta potrebbe essere quella di alternare più strategie di esplorazione lungo la durata delle prove. In tutti gli esperimenti sinora svolti, infatti, la strategia di esplorazione del robot è stata fissata dall'inizio e non è stata più cambiata durante l'esperimento.

Una seconda direzione di ricerca potrebbe prevedere l'introduzione del tempo come variabile da tenere in considerazione. Alcune particolari situazioni USAR potrebbero essere molto pericolose, al punto da richiedere che la maggior parte dell'area venga mappata nei primi minuti della fase di esplorazione. In questo caso occorrerebbe studiare come far variare la strategia di esplorazione in funzione del tempo, delegando agli agenti robotici il compito di riconoscere le condizioni che potrebbero portare ad un cambiamento della strategia scelta. Ad esempio, potrebbe essere utile che i robot adottino, in principio, una strategia di esplorazione tale da ottenere una mappa generale dell'ambiente e che, solo successivamente, tornino sui loro passi per generare una rappresentazione molto più accurata.

Una terza proposta di sviluppo potrebbe indirizzarsi verso lo studio più approfondito dei parametri utilizzati per l'esplorazione di ambienti interni al fine di massimizzare le prestazioni ottenibili con l'algoritmo da noi implementato.

Un aspetto del nostro lavoro di tesi che può essere esteso, è infine la possibilità di esplorare l'ambiente con una squadra di agenti robotici in grado di collaborare tra loro per giungere allo scopo. Questa estensione porterebbe all'utilizzo di un modulo di coordinamento per la gestione dei movimenti dei robot autonomi, rendendo molto più complessa la logica del sistema che dovrebbe occuparsi della comunicazione e della cooperazione tra gli agenti

autonomi che necessitano di interagire tra loro per scambiarsi informazioni e segnali e per collaborare nel compito di esplorazione.

Bibliografia

[1] S. Amarjeet, A. Krause, C. Guestrin, and W. J. Kaiser. Efficient informative sensing using multiple robots. *Journal of Artificial Intelligence Research (JAIR)*, 34:707–755, 2009.

[2] F. Amigoni. Experimental evaluation of some exploration strategies for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2818–2823, 2008.

[3] F. Amigoni, N. Basilico, and A. Quattrini Li. How much worth is coordination of mobile robots for exploration in search and rescue? In *Proceedings of the RoboCup International Symposium*, pages 106–117, 2012.

[4] F. Amigoni, A. Caltieri, R. Cipolleschi, G. Conconi, M. Giusto, M. Luperto, and M. Mazuran. PoAReT Team Description Paper (http://home.deib.polimi.it/amigoni/research/PoAReT-tdp-2012.pdf). *RoboCup2012 CD*, 2012.

[5] F. Amigoni and V. Schiaffonati. Autonomous mobile robots as technical artifacts: A discussion of experimental issues. In *Model-Based Reasoning in Science and Technology*, pages 527–542. 2014.

[6] B. Balaguer, S. Balakirsky, S. Carpin, M. Lewis, and C. Scrapper. USARSim: a validated simulator for research in robotics and automation. *Workshop on "Robot Simulator: Available Software, Scientific Application, and Future Trends" at IROS2008*, 2008.

[7] S. Balakirsky. USARSim: Providing a framework for multi-robot performance evaluation. In *Proceedings of Performance Metrics for Intelligent Systems*, pages 98–102, 2006.

[8] N. Basilico and F. Amigoni. Exploration strategies based on multi-criteria decision making for searching environments in rescue operations. *Autonomous Robots*, 31(4):401–417, 2011.

[9] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper. USARSim: a robot simulator for research and education. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1400–1405, 2007.

[10] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *The Journal of the ACM (JACM)*, 32(3):505–536, 1985.

[11] H. H. González-Baños and J. Latombe. Navigation strategies for exploring indoor environments. *The International Journal of Robotics Research*, 21(10-11):829–848, 2002.

[12] M. Grabisch and C. Labreuche. A decade of application of the Choquet and Sugeno integrals in multi-criteria decision aid. *4OR-A Quarterly Journal of Operations Research*, 6(1):1–44, 2008.

[13] L. E. Kavraki, M. N. Kolountzakis, and J. Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14(1):166–171, 1998.

[14] L. E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[15] S. M. LaValle. *Planning algorithms.* Cambridge University Press, 2006.

[16] B. Reese. AlphA*: An ϵ-admissible Heuristic Search Algorithm (http://home1.stofanet.dk/breese/papers.html). 1999.

[17] C. Stachniss and W. Burgard. Exploring unknown environments with mobile robots using coverage maps. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 1127–1134, 2003.

[18] B. Tovar, L. Muñoz Gómez, R. Murrieta-Cid, M. Alencastre-Miranda, R. Monroy, and S. Hutchinson. Planning exploration strategies for simultaneous localization and mapping. *Robotics and Autonomous Systems*, 54(4):314–331, 2006.

[19] A. Visser and B. Slamet. Including communication success in the estimation of information gain for multi-robot exploration. In *Proceedings of the International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, pages 680–687, 2008.

[20] A. Visser, Xingrui-Ji, M. van Ittersum, L. González J., and L. Stancu. Beyond Frontier Exploration. In *RoboCup 2007: Robot Soccer World Cup XI*, volume 5001 of *Lecture Notes in Computer Science*, chapter 10, pages 113–123. 2008.

[21] P. Walker, A. Kolling, and M. Lewis. Human exploration patterns in unknown, time-sensitive environments. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 2870–2876, 2011.

[22] K. Wurm, C. Stachniss, and W. Burgard. Coordinated multi-robot exploration using a segmentation of the environment. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 1160–1165, 2008.

[23] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings of the Computational Intelligence in Robotics and Automation (CIRA)*, pages 146–151, 1997.

Appendice A

Aspetti matematici generali

In questa appendice fissiamo i punti affrontati in ambito teorico. Questo "*descrizione formale*", redatta nella fase iniziale, è stato molto utile e ha permesso di semplificare lo svolgimento delle fasi successive della tesi.

- discretizzazione del tempo:
  - il campionamento casuale di punti sulla mappa, da cui il robot dovrà passare, non permette una discretizzazione uniforme del tempo;
  - la rappresentazione degli elementi conosciuti (mappa libera, ostacoli, frontiere...) farà riferimento alla conoscenza del robot in un determinato step del path seguito;
- M^t: rappresentazione della mappa conosciuta fino all'istante di tempo t;
  - $F^t \subseteq M^t$: area della mappa identificata come libera al tempo t;
  - $O^t \subseteq M^t$: area della mappa identificata come occupata da ostacoli al tempo t;
  - $G^t \subseteq M^t$: insieme delle frontiere conosciute nella mappa al tempo t;
    dall'implementazione esistente si può notare che la rappresentazione dell'ambiente è in segmenti e le frontiere sono rappresentate come segmenti/linee (possiedono un punto di inizio ed un punto di fine);
  - dove: $F^t \cap O^t = \emptyset$;
  - dove: $G^t \cap O^t = \emptyset$;
  - dove: $G^t \cap F^t = \emptyset$: le frontiere sono rappresentate come elementi posizionati tra area conosciuta e sconosciuta della mappa;

  - F^t, O^t, G^t: sono tutti disgiunti a coppie;

- r^t: posizione del robot al tempo t: $r^t \in F^t$: il robot deve essere nell'area libera della mappa;
- la destinazione verrà scelta tra l'insieme delle frontiere G^t (ogni frontiera viene identificata da un singolo punto: il centro del segmento che rappresenta la frontiera stessa) valutando l'utilità di diversi path per raggiungerla;
- $\mathcal{P}^t$: insieme infinito di tutti i possibili path da r^t ad un qualsiasi punto della mappa libera al tempo t F^t;
- $\bar{\mathcal{P}}^t$: sottoinsieme finito dei possibili path da r^t ad una destinazione $g^t \in G^t$ passando per i punti campionati sulla mappa libera al tempo t F^t; $\bar{\mathcal{P}}^t \subset \mathcal{P}^t$;
- p^t: possibile path verso una destinazione; $p^t \in \bar{\mathcal{P}}^t$;
- $f(p^t)$: funzione di utilità per la valutazione dei path;
- p^{*t}: path migliore al tempo t che collega la posizione attuale del robot con una destinazione; $p^{*t} = \underset{p^t \in \bar{\mathcal{P}}^t}{argmax} f(p^t)$: path che massimizza la funzione di utilità; $p^{*t} \in \bar{\mathcal{P}}^t$;
- movimento del robot e aggiornamento della mappa: $M^t \rightarrow M^{t+1}$: facciamo riferimento mappa al tempo t+1.

Appendice B

Algoritmo A^*

Cosa fare dopo aver ricavato un grafo di punti casuali campionati sulla porzione libera di una mappa? La risposta a questa domanda è la necessità di estrarre i cammini più interessanti che portano a delle possibili destinazioni. Ma come? A questa domanda dà invece risposta l'algoritmo A^* [10] (Pronunciato "A Star" in inglese). A^* è un algoritmo di ricerca applicato a grafi, non orientati e con pesi sugli archi non negativi, che individua un percorso da un dato nodo iniziale verso un dato nodo destinazione (o che passi un test di goal dato). Questo algoritmo utilizza una "stima euristica" che classifica ogni nodo attraverso una stima della strada migliore che passa attraverso tale nodo. A* è anche un esempio di ricerca best-first (letteralmente ricerca prima il migliore).

$$f(x) = g(x) + h(x) \tag{B.0.1}$$

dove:

- $g(x)$: è il costo per raggiungere il nodo x partendo dall'origine, calcolato come somma delle lunghezze degli archi attraversati;
- $h(x)$: euristica che stima il costo per raggiungere la destinazione partendo dal nodo x.

Ad ogni iterazione dell'algoritmo viene scelto il nodo (tra i nodi generati e non ancora scelti) con il minore valore di $f(x)$, fino ad arrivare alla destinazione.

Attraverso l'uso di questo algoritmo, per trovare una soluzione ottima l'euristica $h()$ deve essere:

- ammissibile:

$$h(x) \leq C(x) \, \forall x \tag{B.0.2}$$

- consistente (solo se memorizzo ed elimino i nodi corrispondenti a posizioni già visitate):

$$h(x) \leq c(x,y) + h(y) \, \forall \, \text{arco}\,(x,y) \tag{B.0.3}$$

Algoritmo B.1 Algoritmo A^* implementato in pseudo-codice.

```
List<Node> A*(start,goal){
  //Insieme dei nodi già valutati
  closedset = List();
  //Insieme di nodi che possono essere valutati
  openset = List();
  openSet.append(start);
  //Distanza dalla partenza seguendo il percorso ottimale
  start.setGValue(0);
  //La mappa dei nodi selezionati
  openSet = List();
  start.setHValue(heuristicDistance(start, goal));
  //Stima della distanza totale dall'inizio fino alla destinazione
  start.setFValue(start.getHValue());
  while(!openSet->isEmpty()){
    x = chooseNodeWithLowestFValue(openSet);
    if (x == goal)
      return reconstructPath(openSet,goal);
    openSet->remove(x);
    closedSet->append(x);
    foreach(y, getNeighborNodes(x)){
      if (closedset.contains(y))
        continue;
      tempGValue = x.getGValue() + distance(x,y);
      if(not(openSet.contains(y))){
        openSet->append(y);
        tempIsBetter = true;
      }
      elseif tempGValue < y.getGValue();
        tempIsBetter = true;
```

```
      else
        tempIsBetter = false;
      if (tempIsBetter == true){
        y.setParent(x);
        y.setGValue(tempGValue);
        y.setHValue(heuristicDistance(y,goal);
        y.setFValue(y.getGValue()+y.getHValue());
      }
    }
  }
  return 1;
}

List<Node> reconstructPath(openSet,current)
  if (current.getParent() != NULL){
  p = reconstructPath(openSet, current.getParent());
    return (p + current);
  }
  else
    return the List();
}
```